FEASIBILITY OF A BLAST WAVE ATTENUATION STRUCTURE

by

Dale Richard Hartmann

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Mechanical Engineering

University of Washington

1998

Approved by _____
Chairman of Supervisory Committee

_____

_____

_____

Program Authorized
to Offer Degree Mechanical Engineering

Date _____

Master's Thesis

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Any other reproduction for any purposes or by any means shall not be allowed without my written permission.

Signature _____

Date _____

University of Washington

Abstract

FEASIBILTY OF A BLAST WAVE ATTENUATION STRUCTURE

by Dale Richard Hartmann

Chairman of the Supervisory Committee: Professor Ashley F. Emery
Department of Mechanical Engineering

This thesis begins with an overview of bombings in the United States, followed by the introduction of the Rankine-Hugoniot equations for blast wave pressure. The subsequent chapters develop the one-dimensional and two-dimensional Euler equations. These equations are the solved using the MacCormack finite difference algorithm. The basis of the investigation then begins by placing pole, shear plate and wedge obstacles in the path of the blast wave. The results of these simulations are interpreted and conclusions presented. Finally a synopsis of the existing results and cost analysis for structure hardening are presented.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

The inspiration for this thesis was developed over the course of two years. While at an assignment with the Defense Nuclear Agency, I was privileged to participate in several tests involving car bombs. I became very interested with the effects of the blast waves on different types of structures. After familiarizing myself with the current methods of protecting an existing structure from blast waves, I thought there must be a better way. This thesis is my investigation regarding whether there is a better way to protect existing structures.

ACKNOWLEDGMENTS

The author wishes to acknowledge the support of his wife Gina and sons Sean and Aaron. He also wishes to thank Doctors Emery, Rheinhal, Hyman, Riley, and Fabien of the University of Washington Department of Mechanical Engineering and Doctor Eberhardt of the University of Washington Department of Aeronautical and Astronautical Engineering.

DEDICATION

The author wishes to dedicate this thesis to his wife Gina and sons Sean and Aaron.

INTRODUCTION

BOMBS: THE UNKNOWN MENACE

The threat of terrorist bombings is present every year. Although the United States has been relatively immune from terrorist bombings such as those in Ireland or Israel, 2577 bombings occurred in the United States in 1995 alone. Bombings involving improvised explosive devices such as pipe bombs numbered 1,562 in 1995. The majority of the bombings in 1995 were small in size but produced $105 million damage and 937 casualties (193 killed and 744 injured). The worst bombing in the history of the United States, the Murrah building in Oklahoma City, accounted for $100 million damage and 786 casualties (168 killed and 518 injured).

A cursory review of the above facts and events leads to the question: How can buildings that are possible terrorist bombing targets be protected from the effects of bomb blasts? That question is the basis of this thesis.

CHAPTER 1: THE PROBLEM

TYPES OF EXPLOSIVES

Explosives classified as either: High explosives or Low explosives. High explosives possess certain characteristics that are vastly different than Low explosives. High explosives detonate, which means that when they are initiated a shock or blast wave will form. They also will burst or shatter materials near them, are capable of penetrating materials, and have the capability of lifting or moving objects.

Low explosives, in contrast, do not detonate but burn very rapidly. Since Low explosives do not detonate, the pressure rise that is produced is usually smaller in amplitude but longer in duration than that of a High explosive. This combination tends not to produce an impulse type of blast wave but a slightly 'softer' shock to materials nearby.

BLAST WAVE CHARACTERISTICS

A blast wave generated as the result of the initiation of a contained High explosive is created in the following sequence of events. First hot gases with temperatures of the order 3000 degrees C are generated. In concert with this temperature rise the pressure of these gases is of the order 100 to 300 kilobars. It is this second characteristic of high explosive initiation that is of concern here.

This hot high-pressure gas then expands into the surrounding atmosphere. As expansion occurs, the air surrounding the expanding gas is forced outward. Since air is a compressible medium, a layer of air adjacent to the outer edge of the expanding gas is compressed. It is this layer of compressed air, which is called a blast wave. As the hot gas expands and cools, its pressure falls. The pressure of the layer of compressed air also falls as it is pushed farther outward from the point of detonation. As the gas continues to

cool and expand, at some point in time and space the pressure will fall below atmospheric pressure due to the momentum of the layer of compressed air. This slight negative pressure is called overexpansion and results in a negative phase of the blast wave whereby the outward flow of gases and air is reversed. Eventually after some number of oscillations, equilibrium is reached and the motion of the air stops.

The speed at which the blast wave travels is different depending on the medium in which it is traveling. For a bomb resting on the ground, two waves will exist: one traveling through the ground and another traveling through the air. The latter is what this thesis is concerned with.

The magnitude of the impulse of an explosive blast wave is dependent upon many factors. Some of these factors are:

Is the explosive cased?

What type of explosive is used?

The quantity of explosive used?

What weather conditions are present at the time of detonation?

What is the terrain?

Are any structures nearby?

PROTECTING EXISTING STRUCTURES

THE TRADITIONAL APPROACH

The traditional approach to protecting an existing structure from explosive blast damage is to harden it, that is to increase the structural strength of single or multiple components. Hardening of a building consists of several methods.

The first and most rudimentary method is to harden the glazing, if any is present, on the exterior of the structure. Even a relatively small bomb can cause large amounts of injury and death without structurally damaging a building. This carnage is achieved by shattering normal window glass and propelling the glass fragments at high speed throughout the interior of rooms along the outside perimeter of the building, anyone present in the rooms is impaled with large shards of glass. The use of tempered or tempered safety glass can minimize this effect. However, the use of blast curtains (heavy polymer curtains weighted along the lower edge) or the installation of Mylar film with reinforced window frames can eliminate this hazard. The problem is the curtains have to be kept closed at all times to provide protection, thus rendering the window useless as an architectural entity. In addition the Mylar is relatively expensive.

The most complex, and best, method is to harden the entire structure against the effects of explosive blast waves. This generally consists of a major reconstruction of the interior structural elements of a building or encasing the building inside of another more robust exterior structure capable of withstanding the blast wave. This method, although highly effective, is also difficult to design, disruptive to the occupants, time consuming to construct and very expensive.

THE NEW APPROACH

The approach that this thesis investigates is the use of a blast wave attenuation structure. The form of the blast wave attenuation structure would be such that as the blast wave travels through it the energy of the wave is dissipated. Ideally, the energy of the wave would be lowered to such a degree as to produce and overpressure of no more than one half atmosphere. This lowered pressure, although still high enough to cause some damage, is safe for most commercial types of construction.

The forms of the attenuation structure to be investigated include shear plates, a field of poles, and of wedges. The success of a type of structure will depend upon its ability to attenuate the pressure in less than 15 meters, constructability, low cost, and aesthetics.

CHAPTER 2: THE QUESTION

MODELING BLAST WAVES

BLAST WAVE PARAMETERS

The modeling of a blast wave must address two issues. The first is the modeling of the static aspects of the explosion. These static aspects include pressure, density, temperature and shock wave velocity. The second, and more difficult, is the modeling of the dynamics of the blast wave motion and interactions with objects. These dynamic aspects are the same quantities as the static aspects but are concerned with how these quantities change with time and position.

STATIC ASPECTS

It is important to model the static aspects of an explosion since this is what provides the driving force for the dynamic solutions and simulations. The first static aspect to be modeled is the pressure generated by the detonation of the explosive. The pressure generated is dependent upon several factors:

Type of explosive

Distance from explosive

Position of explosive relative to the ground

The type of explosive is important since each different type of explosive contains a different amount of energy per unit mass. These differences are summarized in the table below.

Table 1 Explosive Equivalents

Explosive	Mass Specific Energy Q_x(kJ/kg)	TNT Equivalent (Q_x/Q_{TNT})
RDX (Cyclonite)	5360	1.185
HMX	5680	1.256
Nitroglycerin (Liquid)	6700	1.481
TNT	4520	1.000
Blasting Gelatin	4520	1.000
Nitroglycerin dynamite	2710	0.600
Semtex	5660	1.250
Compound B	5190	1.148

As can be seen from the table above, the method of normalizing the energies of different explosives relative to that contained in TNT has been developed and is universally accepted.

The relationship between the range and the TNT equivalent charge mass is known as the scaled distance and is given by the following equation:

$$Z = \frac{R}{W^{1/3}} \tag{1}$$

Where W, the universal symbol for TNT equivalents, denotes the mass of explosive and R denotes the range from the detonation of the explosive to the point of interest. Rankine and Hugoniot produced the first analytic solutions for blast wave front parameters in 1870 for normal shocks in ideal gases. These solutions were later expanded by Brode to determine the peak static overpressure in the near and medium fields of the blast wave. The near field is the region where the overpressure is higher than 10 bar (10^6 Pa). The medium field is the region where the overpressure is less than 10 bar (10^6 Pa) and higher than 0.1 bar (10^4 Pa). The equation for the overpressure in the near field is given by

$$p_s = \left(\frac{6.7}{Z^3} + 1\right)(100000) \text{ (Pa)} \tag{2}$$

The equation for the overpressure in the medium field is given by

$$p_s = \left(\frac{0.975}{Z} + \frac{1.455}{Z^2} + \frac{5.85}{Z^3} - 0.019\right)(100000) \text{ (Pa)} \tag{3}$$

The absolute pressure of the detonation is given by

$$p_{absolute} = p_s + p_{ambient} \tag{4}$$

The equations developed by Rankine, Hugoniot and Brode, shown above, dealt with shock waves in free air. These shock waves were allowed to propagate in all directions, or, more simply, in a spherical manner. Since I am concerned with terrorist bombings, the majority of which involve explosives placed on or near the ground, a correction factor is needed on $p_{absolute}$ to account for the reflection from the ground. If the ground were an ideal surface for reflection this correction factor would by 2. However, the ground will respond in two ways that are not ideal. The ground underneath the explosion will compress. The area around the explosion will undergo a fracturing process, which will result in particles being ejected into the air. There is no analytical method of quantifying these responses, but empirical evidence suggests a correction factor given by

$$p_a = 1.8 p_{absolute} \tag{5}$$

This correction factor is applied to the pressure calculations for an explosion to provide the input pressure for all analytical and finite difference calculations. The Mach number of the detonation shock wave is given by

$$M_x = \left(\left(\frac{p_a}{p_{ambient}} - 1\right)\left(\frac{\gamma + 1}{2\gamma + 1}\right)\right)^{1/2} \tag{6}$$

The units in the equations below are correct for p_a expressed in Pa.

The density of the gas behind the detonation shockwave is given by

$$\rho_2 = \frac{\rho_{ambient}}{\left(1-\frac{2}{\gamma+1}\right)\left(1-\frac{1}{M_x{}^2}\right)} \text{ (Kg/m}^3\text{)} \tag{7}$$

The velocity of the detonation shockwave is given by

$$u_2 = c_1\left(\frac{2}{\gamma+1}\left(M_x - \frac{1}{M_x}\right)\right) \text{ (m/s)} \tag{8}$$

Where c_1 is the ambient speed of sound

$$c_1 = \left(\gamma \frac{p_{ambient}}{\rho_{ambient}}\right)^{1/2} \text{ (m/s)} \tag{9}$$

The temperature of the gas behind the detonation shockwave is given by

$$T_2 = \frac{p_a}{\rho_2 R} \text{ (K)} \tag{10}$$

Where

$$R = \frac{R_u}{MW_{air}} \text{(J/kg K)} \tag{11}$$

With

$$R_u = 8314.51 \text{(J/kmole K)} \tag{12}$$

$$MW_{air} = 29 \text{ (Kg/kmole)} \tag{13}$$

The speed of sound behind the detonation shockwave is given by

$$c_2 = \left(c_1{}^2 \frac{T_2}{T_{ambient}} \right)^{1/2} \text{ (m/s)} \tag{14}$$

DYNAMIC ASPECTS

THE EULER EQUATIONS

Air has a small thermal conductivity (κ) and also a small viscosity (υ) therefore the terms in the Navier-Stokes equations that depend on κ and υ can be neglected, which leads to equations in which the convective terms dominate and the fluid (in this case air) is treated as inviscid. The inviscid 2-D Navier-Stokes equations are called the Euler equations and are the equations dealt with in this paper. These equations are much simpler in form and also programming effort.

The Euler equations are written as

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0 \tag{15}$$

where

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix} \qquad F = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(E+p) \end{bmatrix} \qquad G = \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(E+p) \end{bmatrix} \tag{16}$$

and

$$E = \rho\left(e + \frac{1}{2}\left(u^2 + v^2\right)\right) \tag{17}$$

The Euler equations represent a system of equations that conserve mass, momentum and energy. These equations must be solved as a system to capture nonlinearities such as shocks.

The Euler equations also need an equation of state, which is the ideal gas law.

$$p = \rho RT = (\gamma - 1)\left(E - \frac{1}{2}\rho\left(u^2 + v^2\right)\right) \tag{18}$$

CHAPTER 3: ONE DIMENSIONAL EQUATIONS

THE 1-D EULER EQUATIONS

The first step in modeling the blast wave is to construct a one-dimensional model with time and distance as the parameters. The 1-D Euler equations are written as

$$\frac{\partial Q}{\partial t}+\frac{\partial F}{\partial x}=0 \tag{19}$$

where

$$Q=\begin{bmatrix}\rho\\ \rho u\\ E\end{bmatrix} \qquad F=\begin{bmatrix}\rho u\\ \rho u^2+p\\ u(E+p)\end{bmatrix} \tag{20}$$

and

$$E=\rho\left(e+\frac{1}{2}\left(u^2\right)\right) \tag{21}$$

The equation of state is

$$p=\rho RT=(\gamma-1)\left(E-\frac{1}{2}\rho\left(u^2\right)\right) \tag{22}$$

This system of equations yields the following three equations

Conservation of mass

$$\frac{\partial \rho}{\partial t}+\frac{\partial(\rho u)}{\partial x}=0 \tag{23}$$

Conservation of momentum

$$\frac{\partial(\rho u)}{\partial t}+\frac{\partial(\rho u^2+p)}{\partial x}=0 \tag{24}$$

Conservation of energy

$$\frac{\partial E}{\partial t}+\frac{\partial u(E+p)}{\partial x}=0 \tag{25}$$

FINITE DIFFERENCING

To evaluate this system of equations the method of finite differencing is used. Finite differencing is a powerful numerical technique that can be used to solve partial differential equations. Finite differencing algorithms come in two varieties: explicit and implicit. An explicit algorithm solves each equation in turn and then applies those results to the initial values for the next time step. An implicit algorithm solves the system of equations simultaneously via matrix manipulations. I chose the explicit form of finite differencing instead of the implicit form for the following reasons:

Ease of coding

Lower computational requirements

Ability to run on desktop computing systems

The first step in constructing a finite differencing program is to choose an algorithm. Many different algorithms exist but to be useful for the problem at hand the chosen algorithm must be able to "capture" the shock discontinuities well and also handle surface interactions. Several algorithms meet these requirements:

Forward Time Centered Space (FTCS)

MacCormack

Harten-Ye

1st Order Roe

I initially investigated the use of the FTCS algorithm, however I was not pleased with the shock "capturing" abilities of this algorithm. This algorithm smears or spreads the shock discontinuity over 4 to 5 Δx intervals, it also has dispersion errors which appear as oscillations at the shock discontinuity. The following figure demonstrates these points.

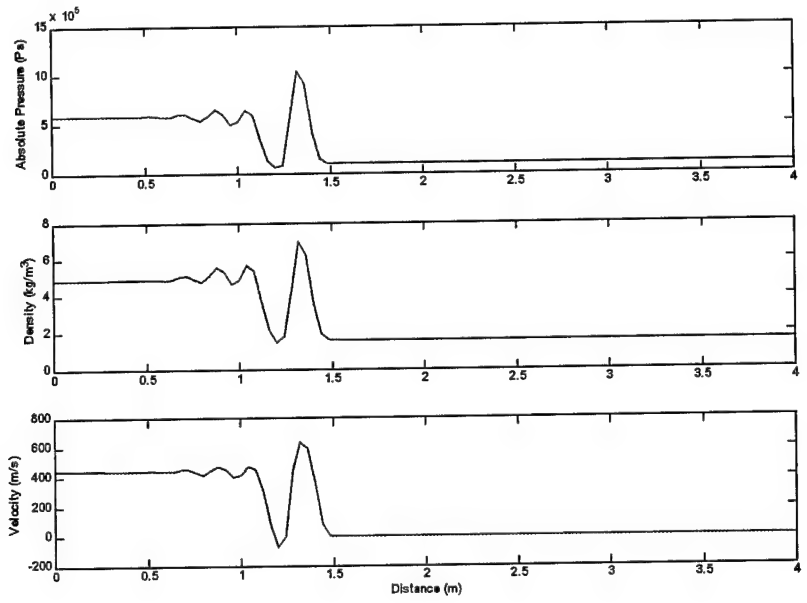

Figure 1 FTCS Output

The next algorithm I investigated was MacCormack's. This algorithm is a two-step method that is known for "capturing" shock discontinuities very well. In contrast to the FTCS algorithm, MacCormack's does not have the dispersion errors at the shock discontinuity. The following figure demonstrates these points.

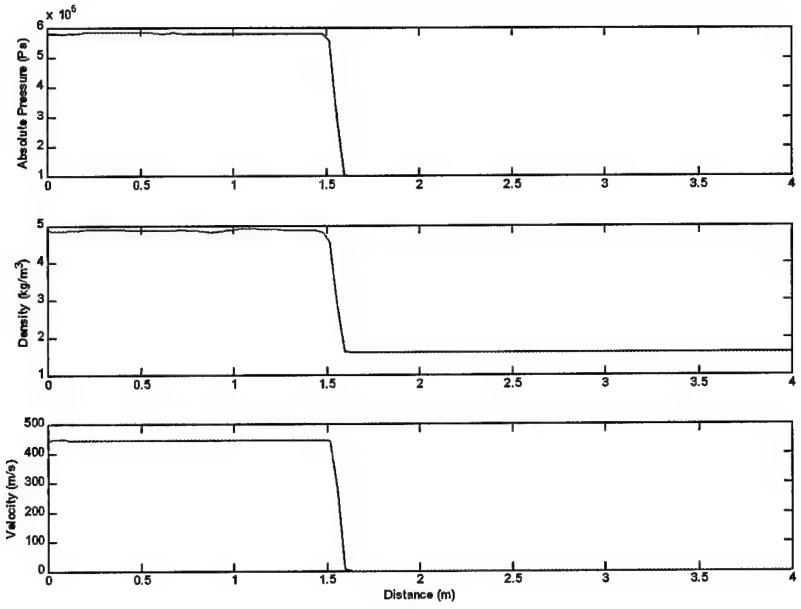

Figure 2 MacCormack Output

The Harten-Yee and 1st Order Roe methods were abandoned early on due to their complexity and coding difficulties and because they have been shown not to be substantially superior to MacCormack's method.

Due to the simplicity and shock capturing abilities of the MacCormack algorithm, I have chosen to use it as the basis for the remainder of this thesis.

The method of the MacCormack algorithm is contained in the two steps below:

$$\text{Predictor step: } \overline{Q_j} = Q_j^{\,n} - \Delta t \Delta_x F_j^{\,n} \tag{26}$$

$$\text{Corrector step: } Q_j^{\,n+1} = \frac{1}{2}\left(\overline{Q_j} + Q_j^{\,n}\right) - \frac{\Delta t}{2}\nabla_x \overline{F_j} \tag{27}$$

where

$$\Delta_x = F_{j+1}{}^n - F_j{}^n \tag{28}$$

$$\nabla_x = \overline{F_j} - \overline{F_{j-1}} \tag{29}$$

For the one dimensional case the MacCormack algorithm equations become

Conservation of mass

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} = 0 \tag{30}$$

$$\text{Predictor: } \overline{\rho_j} = \rho_j{}^n - \frac{\Delta t}{\Delta x_x}\left((\rho u)_{j+1}{}^n - (\rho u)_j{}^n \right) \tag{31}$$

$$\text{Corrector: } \rho_j{}^{n+1} = \frac{\overline{\rho_j} - \rho_j{}^n}{2} - \frac{\Delta t}{2\Delta x}\left(\overline{\rho u_j{}^n} - \overline{\rho u_{j-1}{}^n} \right) \tag{32}$$

Conservation of momentum

$$\frac{\partial (\rho u)}{\partial t} + \frac{\partial \left(\rho u^2 + p\right)}{\partial x} = 0 \tag{33}$$

$$\text{Predictor: } \overline{(\rho u)_j} = (\rho u)_j{}^n - \frac{\Delta t}{\Delta x}\left(\frac{\left((\rho u)_{j+1}{}^n\right)^2}{\rho_{j+1}{}^n} - \frac{\left((\rho u)_j{}^n\right)^2}{\rho_j{}^n} + p_{j+1}{}^n - p_j{}^n \right) \tag{34}$$

$$\text{Corrector: } (\rho u)_j{}^{n+1} = \frac{\overline{(\rho u)_j} - (\rho u)_j{}^n}{2} - \frac{\Delta t}{2\Delta x}\left(\overline{(\rho u)_j} - \overline{(\rho u)}_{j-1} + \overline{p}_j - \overline{p}_{j-1} \right) \tag{35}$$

Conservation of energy

$$\frac{\partial E}{\partial t}+\frac{\partial u(E+p)}{\partial x}=0 \tag{36}$$

$$\text{Predictor: } \overline{E_j} = E_j^{\,n} - \frac{\Delta t}{\Delta x}\left(\frac{(\rho u)_{j+1}^{\,n}}{\rho_{j+1}^{\,n}}\left(E_{j+1}^{\,n}-p_{j+1}^{\,n}\right)-\frac{(\rho u)_j^{\,n}}{\rho_j^{\,n}}\left(E_j^{\,n}-p_j^{\,n}\right)\right) \tag{37}$$

$$\text{Corrector: } E_j^{\,n+1} = \frac{\overline{E_j}+E_j^{\,n}}{2} - \frac{\Delta t}{2\Delta x}\left(\overline{E}_j-\overline{E}_{j-1}+\overline{p}_j-\overline{p}_{j-1}\right) \tag{38}$$

The equations for the conservation of momentum and energy above contain a pressure term that is evaluated using the equation of state. The equation of state is the ideal gas law and is given by

$$p=(\gamma-1)\left(E-\frac{1}{2}\frac{(\rho u)^2}{\rho}\right) \tag{39}$$

In the above form the equation state can be evaluated using the results of the MacCormack algorithm.

The final concern for any finite differencing algorithm is stability. The use of the Courant-Friedrichs-Lewy or CFL number is the most widely used method of controlling stability. The CFL number arises out of the results from a Von-Neumann stability analysis. The theory of Von-Neumann utilizes a Fourier transform to transform the finite difference solution into a wave space solution. The amplitudes of the waves in this resulting solution will grow or decay based upon the particular finite difference algorithm chosen and the value of the CFL number. The waves in the Von-Neumann solution that grow are unstable, those that decay are stable. The results of the Von-Neumann analysis require that the CFL number be less than or equal to one. The physical implication of the CFL number being less than or equal to one is that the sum of the amplitudes of the waves that decay is larger than the sum of the amplitude of the waves that grow. Thus if

the Courant number remains below the value of 1 the decaying waves will dominate and the algorithm will remain stable.

$$CFL = \frac{\Delta x}{(|u| + c)\Delta t} \tag{40}$$

where c is the speed of sound for the given pressure and density and CFL is the Courant-Friedrichs-Lewy number. To determine the time step used in the finite differencing code the CFL equation above is manipulated such that the following equation for a one-dimensional algorithm time step results.

$$\Delta t = CFL\frac{\Delta x}{(|u| + c)} \tag{41}$$

VALIDATING THE ALGORITHM

At this point, after developing the equations for the one-dimensional case and constructing the finite difference algorithm, validation of the method is appropriate. To validate the code a model was constructed with a right traveling shock that reflects from an infinite wall at the right most-boundary. I selected this validation method due to its ease of coding and readily available analytic solution.

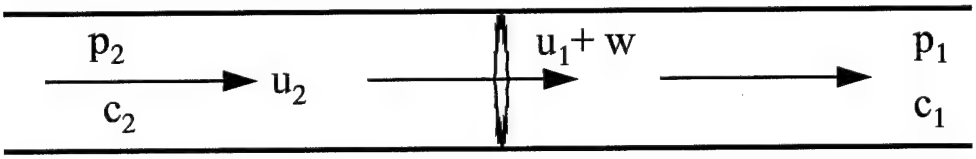

Figure 3 Shocktube with right traveling wave

The actual validation was done by allowing the finite difference solution to run long enough following the interaction with the right most wall and then comparing the

pressure and density values from the model to the analytic values calculated from the analytic solution (Shapiro)

The constant initial conditions in the program code are

Table 2 Fixed initial conditions

Temperature	Gamma	Density	Pressure	Speed of sound
298 K	1.4	1.614 kg/m^3	101325 Pa	296 m/s

The initial conditions input to the finite difference model were

Table 3 Input initial conditions

Courant number	Calculation distance	Number of position steps	Number of time steps	Distance from charge	Explosive	Explosive mass
0.5	4 m	101	250	10 m	TNT	200 kg

The program calculated values behind the shock prior to the wall interface were determined using the Rankine-Hugoniot and Brode equations, equations 2, 3, 5, 6, 7, 8, 9, 10 and 14. The values output from the program are tabulated below.

Table 4 Calculated free stream shock values

Static pressure of blast	5.7417×10^5 Pa
Mach number of shock	2.2504
Density	4.8729 kg/m^3
Shock velocity	446.18 m/s
Temperature	410.97 K
Speed of sound	348.97 m/s

The calculated values from the program behind the shock after the wall interface were

Table 5 Calculated reflection shock values

Static pressure of blast	2.246×10^6 Pa
Density	12.179 kg/m^3

To obtain the analytic values an iterative method was used. Following the reflection from the right most wall the shock is now traveling to the left. The analytic solution is to hold the shock stationary and iterate for the value of W.

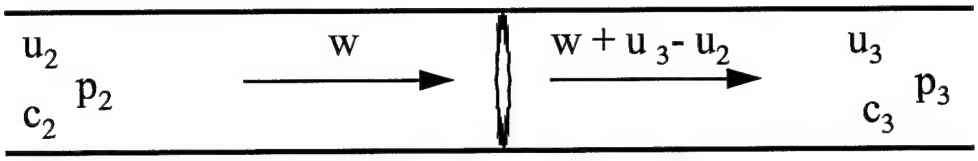

Figure 4 Shocktube with stationary wave

The equations required for the iterative solutions are

$$M_x = \frac{w + u_2}{c_2} \tag{42}$$

$$\frac{p_3}{p_2} = 1 + \frac{2\gamma}{\gamma + 1}\left(M_x^{\,2} - 1\right) \tag{43}$$

$$\frac{\rho_3}{\rho_2} = \frac{1}{1 - \frac{2}{\gamma + 1}\left(1 - \frac{1}{M_x^{\,2}}\right)} \tag{44}$$

$$u_3 - u_2 = c_2\left(\frac{2}{\gamma + 1}\left(M_x - \frac{1}{M_x}\right)\right) \tag{45}$$

Using the equations above, equations 41 through 44, with the initial values listed in the table below, the results of the analytic solution will be those listed in the table below.

Table 6 Analytic results for shocktube

Initial values	u_2	c_2	p_2	ρ_2	γ
	446.18 m/s	348.1525 m/s	574.17 kPa	4.8729 kg/m^3	1.4
Calculated values	W		P_3	ρ_3	
	180.5 m/s		2.0747 MPa	11.4963 kg/m^3	

Comparing these analytic values with the finite difference calculated values yields an error of +6% for the density and +8% for the pressure. Which means that the finite difference program will yield a slightly conservative (higher than true values) result. This is acceptable for my purposes.

CHAPTER 4: TWO-DIMENSIONAL EQUATIONS

THE 2-D EULER EQUATIONS

The addition of a second dimension to the 1-D Euler equations is fairly straightforward. The second dimension requires one additional equation to the matrix and two additional terms. The 2-D Euler equations are written

$$\frac{\partial Q}{\partial t}+\frac{\partial F}{\partial x}+\frac{\partial G}{\partial y}=0 \tag{46}$$

where

$$Q=\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ E\end{bmatrix} \quad F=\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ u(E+p)\end{bmatrix} \quad G=\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ v(E+p)\end{bmatrix} \tag{47}$$

and

$$E=\rho\left(e+\frac{1}{2}\left(u^2+v^2\right)\right) \tag{48}$$

The equation of state is

$$p=\rho RT=(\gamma-1)\left(E-\frac{1}{2}\rho\left(u^2+v^2\right)\right) \tag{49}$$

This system of equations yields the following four equations

Conservation of mass

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} + \frac{\partial (\rho v)}{\partial y} = 0 \tag{50}$$

Conservation of x momentum

$$\frac{\partial (\rho u)}{\partial t} + \frac{\partial \left(\rho u^2 + p\right)}{\partial x} + \frac{\partial (\rho uv)}{\partial y} = 0 \tag{51}$$

Conservation of y momentum

$$\frac{\partial (\rho u)}{\partial t} + \frac{\partial (\rho uv)}{\partial x} + \frac{\partial \left(\rho v^2 + p\right)}{\partial y} = 0 \tag{52}$$

Conservation of energy

$$\frac{\partial E}{\partial t} + \frac{\partial u(E + p)}{\partial x} + \frac{\partial v(E + p)}{\partial y} = 0 \tag{53}$$

The method of the MacCormack algorithm for two dimensions is similar to that for one dimension. The two dimensional algorithm is contained in the two steps below

$$\text{Predictor step: } \overline{Q_{j,k}} = Q_{j,k}{}^{n} - \Delta t\left(\Delta_x F_{j,k}{}^{n} + \Delta_y G_{j,k}{}^{n}\right) \tag{54}$$

$$\text{Corrector step: } Q_{j,k}{}^{n+1} = \frac{1}{2}\left(\overline{Q_{j,k}} + Q_{j,k}{}^{n}\right) - \frac{\Delta t}{2}\left(\nabla_x \overline{F_{j,k}} + \nabla_y \overline{G_{j,k}}\right) \tag{55}$$

where

$$\Delta_x = F_{j+1,k}{}^{n} - F_{j,k}{}^{n} \tag{56}$$

$$\nabla_x = \overline{F_{j,k}} - \overline{F_{j-1,k}} \tag{57}$$

and

$$\Delta_y = G_{j,k+1}{}^n - G_{j,k}{}^n \tag{58}$$

$$\nabla_y = \overline{G_{j,k}} - \overline{G_{j,k-1}} \tag{59}$$

For the two dimensional case the MacCormack algorithm equations become

Conservation of mass

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} + \frac{\partial (\rho v)}{\partial y} = 0 \tag{60}$$

$$\text{Predictor: } \overline{\rho_{j,k}} = \rho_{j,k}{}^n - \Delta t \left(\frac{(\rho u)_{j+1,k}{}^n - (\rho u)_{j,k}{}^n}{\Delta x} + \frac{(\rho v)_{j,k+1}{}^n - (\rho v)_{j,k}{}^n}{\Delta y} \right) \tag{61}$$

$$\text{Corrector: } \rho_{j,k}{}^{n+1} = \frac{\overline{\rho_{j,k}} - \rho_{j,k}{}^n}{2} - \frac{\Delta t}{2} \left(\frac{\overline{\rho u_{j,k}{}^n} - \overline{\rho u_{j-1,k}{}^n}}{\Delta x} + \frac{\overline{\rho v_{j,k}{}^n} - \overline{\rho v_{j,k-1}{}^n}}{\Delta y} \right) \tag{62}$$

Conservation of x momentum

$$\frac{\partial (\rho u)}{\partial t} + \frac{\partial (\rho u^2 + p)}{\partial x} + \frac{\partial (\rho u v)}{\partial y} = 0 \tag{63}$$

$$\left(\overline{\rho u}\right)_{j,k} = \left(\rho u\right)_{j,k}{}^{n}$$

$$\text{Predictor: } -\Delta t\left[\frac{1}{\Delta x}\left(\frac{\left(\left(\rho u\right)_{j+1,k}{}^{n}\right)^2}{\rho_{j+1,k}{}^{n}} - \frac{\left(\left(\rho u\right)_{j,k}{}^{n}\right)^2}{\rho_{j,k}{}^{n}} + p_{j+1,k}{}^{n} - p_{j,k}{}^{n}\right)\right. \tag{64}$$

$$\left. + \frac{1}{\Delta y}\left(\frac{\left(\rho u\right)_{j,k+1}{}^{n}\left(\rho v\right)_{j,k+1}{}^{n}}{\rho_{j,k+1}{}^{n}} - \frac{\left(\rho u\right)_{j,k}{}^{n}\left(\rho v\right)_{j,k}{}^{n}}{\rho_{j,k}{}^{n}}\right)\right]$$

$$\left(\rho u\right)_{j,k}{}^{n+1} = \frac{\left(\overline{\rho u}\right)_{j,k} - \left(\rho u\right)_{j,k}{}^{n}}{2}$$

$$\text{Corrector: } -\frac{\Delta t}{2}\left[\frac{1}{\Delta x}\left(\left(\left(\overline{\rho u}\right)_{j,k}\right)^2 - \left(\left(\overline{\rho u}\right)_{j-1,k}\right)^2 + \overline{p}_{j,k} - \overline{p}_{j-1,k}\right)\right. \tag{65}$$

$$\left. + \frac{1}{\Delta y}\left(\frac{\left(\overline{\rho u}\right)_{j,k}\left(\overline{\rho v}\right)_{j,k}}{\overline{\rho}_{j,k}} - \frac{\left(\overline{\rho u}\right)_{j,k-1}\left(\overline{\rho v}\right)_{j,k-1}}{\overline{\rho}_{j,k-1}}\right)\right]$$

where

$$p = \left(\gamma - 1\right)\left(E - \frac{1}{2}\frac{\left(\rho u\right)^2}{\rho}\right) \tag{66}$$

and

$$\overline{p} = \left(\gamma - 1\right)\left(\overline{E} - \frac{1}{2}\frac{\left(\overline{\rho u}\right)^2}{\overline{\rho}}\right) \tag{67}$$

Conservation of y momentum

$$\frac{\partial(\rho u)}{\partial t}+\frac{\partial(\rho uv)}{\partial x}+\frac{\partial\left(\rho v^2+p\right)}{\partial y}=0 \tag{68}$$

$$\left(\overline{\rho v}\right)_{j,k}=\left(\rho v\right)_{j,k}{}^{n}$$

$$\text{Predictor: } -\Delta t\left[\frac{1}{\Delta x}\left({}^{n}\frac{\left(\rho u\right)_{j,k+1}{}^{n}\left(\rho v\right)_{j,k+1}{}^{n}}{\rho_{j,k+1}{}^{n}}-\frac{\left(\rho u\right)_{j,k}{}^{n}\left(\rho v\right)_{j,k}{}^{n}}{\rho_{j,k}{}^{n}}\right)\right. \tag{69}$$

$$\left.+\frac{1}{\Delta y}\left(\frac{\left(\left(\rho v\right)_{j+1,k}{}^{n}\right)^2}{\rho_{j+1,k}{}^{n}}-\frac{\left(\left(\rho v\right)_{j,k}{}^{n}\right)^2}{\rho_{j,k}{}^{n}}+p_{j+1,k}{}^{n}-p_{j,k}{}^{n}\right)\right]$$

$$\left(\rho v\right)_{j,k}{}^{n+1}=\frac{\left(\overline{\rho v}\right)_{j,k}-\left(\rho v\right)_{j,k}{}^{n}}{2}$$

$$\text{Corrector: } -\frac{\Delta t}{2}\left[\frac{1}{\Delta x}\left(\frac{\left(\overline{\rho u}\right)_{j,k}\left(\overline{\rho v}\right)_{j,k}}{\overline{\rho}_{j,k}}-\frac{\left(\overline{\rho u}\right)_{j-1,k}\left(\overline{\rho v}\right)_{j-1,k}}{\overline{\rho}_{j-1,k}}\right)\right. \tag{70}$$

$$\left.+\frac{1}{\Delta y}\left(\left(\left(\overline{\rho v}\right)_{j,k}\right)^2-\left(\left(\overline{\rho v}\right)_{j,k-1}\right)^2+\overline{p}_{j,k}-\overline{p}_{j,k-1}\right)\right]$$

where

$$p = (\gamma - 1)\left(E - \frac{1}{2}\frac{(\rho v)^2}{\rho} \right) \tag{71}$$

and

$$\overline{p} = (\gamma - 1)\left(\overline{E} - \frac{1}{2}\frac{\left(\overline{\rho v}\right)^2}{\overline{\rho}} \right) \tag{72}$$

Conservation of energy

$$\frac{\partial E}{\partial t} + \frac{\partial u(E+p)}{\partial x} + \frac{\partial v(E+p)}{\partial y} = 0 \tag{73}$$

$$\overline{E_{j,k}} = {E_{j,k}}^n -$$

$$\text{Predictor: } \Delta t\left[\frac{1}{\Delta x}\left(\frac{{(\rho u)_{j+1,k}}^n}{{\rho_{j+1,k}}^n}\left({E_{j+1,k}}^n - {p_{j+1,k}}^n \right) - \frac{{(\rho u)_{j,k}}^n}{{\rho_{j,k}}^n}\left({E_{j,k}}^n - {p_{j,k}}^n \right) \right) \right. \tag{74}$$

$$\left. + \frac{1}{\Delta y}\left(\frac{{(\rho v)_{j,k+1}}^n}{{\rho_{j,k+1}}^n}\left({E_{j,k+1}}^n - {p_{j,k+1}}^n \right) - \frac{{(\rho v)_{j,k}}^n}{{\rho_{j,k}}^n}\left({E_{j,k}}^n - {p_{j,k}}^n \right) \right) \right]$$

$$E_j^{n+1} = \frac{\overline{E_j} - E_j^n}{2}$$

$$\text{Corrector: } -\frac{\Delta t}{2}\left[\frac{1}{\Delta x}\left(\frac{\left(\overline{\rho u}\right)_{j,k}}{\overline{\rho}_{j,k}}\left(\overline{E}_{j,k}+\overline{p}_{j,k}\right)-\frac{\left(\overline{\rho u}\right)_{j-1,k}}{\overline{\rho}_{j-1,k}}\left(\overline{E}_{j-1,k}+\overline{p}_{j-1,k}\right)\right)\right. \tag{75}$$

$$\left.+\frac{1}{\Delta y}\left(\frac{\left(\overline{\rho v}\right)_{j,k}}{\overline{\rho}_{j,k}}\left(\overline{E}_{j,k}+\overline{p}_{j,k}\right)-\frac{\left(\overline{\rho v}\right)_{j,k-1}}{\overline{\rho}_{j,k-1}}\left(\overline{E}_{j,k-1}+\overline{p}_{j,k-1}\right)\right)\right]$$

The pressure terms in the conservation of energy equations above are evaluated by using the equation of state. The pressure terms are distinct for each direction. The x direction pressure term is given by

$$p = (\gamma+1)\left(E - \frac{1}{2}\left(\frac{(\rho u)^2}{\rho}\right)\right) \tag{76}$$

Similarly the y direction pressure term is given by

$$p = (\gamma+1)\left(E - \frac{1}{2}\left(\frac{(\rho v)^2}{\rho}\right)\right) \tag{77}$$

The time step in two dimensions is a more complicated expression given by

$$\Delta t = CFL\frac{1}{\frac{|u|}{\Delta x}+\frac{|v|}{\Delta y}+c\sqrt{\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}}} \tag{78}$$

where c is the speed of sound for the given density and pressure and CFL is the Courant-Friedrichs-Lewy number.

VALIDATING THE ALGORITHM

Applying reasoning similar to the one-dimensional case, an input pressure was applied to the x edge of the computational domain. The effect of this input pressure was the same as the calculations preformed by the one-dimensional code. Inputting the same parameters into both the 2-D and 1-D programs allowed comparison of the output matrices, which were identical. Therefore the 2-D code is validated since the 1-D code has been proved correct and acceptable. Following validation of the 2-D code for a linear pressure wave traveling in the y direction, the same procedure was applied for a wave traveling in the x direction. The same results were obtained.

The results of this validation are shown in the tables below

Table 7 1-D program calculated values

Density				
4.8729	3.0026	1.7713	1.6140	1.6140
Momentum				
2174.2	974.5	79.3	0	0
Energy				
1.9395e6	1.0087e6	0.3144e6	0.2533e6	0.2533e6

The results for a 2-D wave traveling in the x direction are given in the tables below

Table 8 2-D x-traveling wave program calculated values

Density				
4.8729	3.0026	1.7713	1.6140	1.6140
4.8729	3.0026	1.7713	1.6140	1.6140
4.8729	3.0026	1.7713	1.6140	1.6140
4.8729	3.0026	1.7713	1.6140	1.6140
4.8729	3.0026	1.7713	1.6140	1.6140
Momentum				
2174.2	974.5	79.3	0	0
2174.2	974.5	79.3	0	0
2174.2	974.5	79.3	0	0
2174.2	974.5	79.3	0	0
2174.2	974.5	79.3	0	0
Energy				
1.9395e6	1.0067e6	0.3144e6	0.2533e6	0.2533e6
1.9395e6	1.0067e6	0.3144e6	0.2533e6	0.2533e6
1.9395e6	1.0067e6	0.3144e6	0.2533e6	0.2533e6
1.9395e6	1.0067e6	0.3144e6	0.2533e6	0.2533e6
1.9395e6	1.0067e6	0.3144e6	0.2533e6	0.2533e6

The results for a y traveling wave are in the tables below

Table 9 2-D y-traveling wave program calculated values

Density				
4.8729	4.8729	4.8729	4.8729	4.8729
3.0026	3.0026	3.0026	3.0026	3.0026
1.7713	1.7713	1.7713	1.7713	1.7713
1.6140	1.6140	1.6140	1.6140	1.6140
1.6140	1.6140	1.6140	1.6140	1.6140
Momentum				
2174.2	2174.2	2174.2	2174.2	2174.2
974.5	974.5	974.5	974.5	974.5
79.3	79.3	79.3	79.3	79.3
0	0	0	0	0
0	0	0	0	0
Energy				
1.9395e6	1.9395e6	1.9395e6	1.9395e6	1.9395e6
1.0067e6	1.0067e6	1.0067e6	1.0067e6	1.0067e6
0.3144e6	0.3144e6	0.3144e6	0.3144e6	0.3144e6
0.2533e6	0.2533e6	0.2533e6	0.2533e6	0.2533e6
0.2533e6	0.2533e6	0.2533e6	0.2533e6	0.2533e6

The results above were achieved with the following input conditions:

Table 10 2-D validation initial conditions

Time step	x-grid	Range to Charge	Explosive Type	Explosive Mass
5e-4 s	5	10 m	TNT	200 kg

CHAPTER 5: TESTING THE NEW APPROACH

The validation of the 2-D code, presented in the previous chapter, was performed with linear wavefronts traveling in only one direction. While this situation might occur with a very large explosion at a great distance, in general this is an unrealistic case. Therefore, the use of symmetry was applied. The final step in the code development process was to place the explosion in one corner of the computational realm and allow the wave to expand in true 2-D fashion. Placement of the explosive in this corner allows for two planes of symmetry and reduces the number of calculations per time step by a factor of two. This reduction of calculations is also beneficial since it ignores the blast wave that travels away from the obstacles.

The initial parameters for an explosion were placed into one corner of the computational realm and allowed to expand freely with no obstacles present. The results obtained were consistent with the expected outcome. This step was necessary to provide a test of the 2-D code in the true two dimensional environment.

OBSTACLE BOUNDARY CONDITIONS

To obtain meaningful results with obstacles in the path of the wave, a thorough understanding of the correct boundary conditions for the obstacles is needed. The minimum size of an obstacle is limited to a grid of 3 by 3 computational points. This minimum size is necessary to allow for the specification of conditions inside the obstacle. There is no limit on the maximum obstacle size, although it obviously cannot exceed the size of the computational realm. The figure below illustrates a minimum size obstacle.

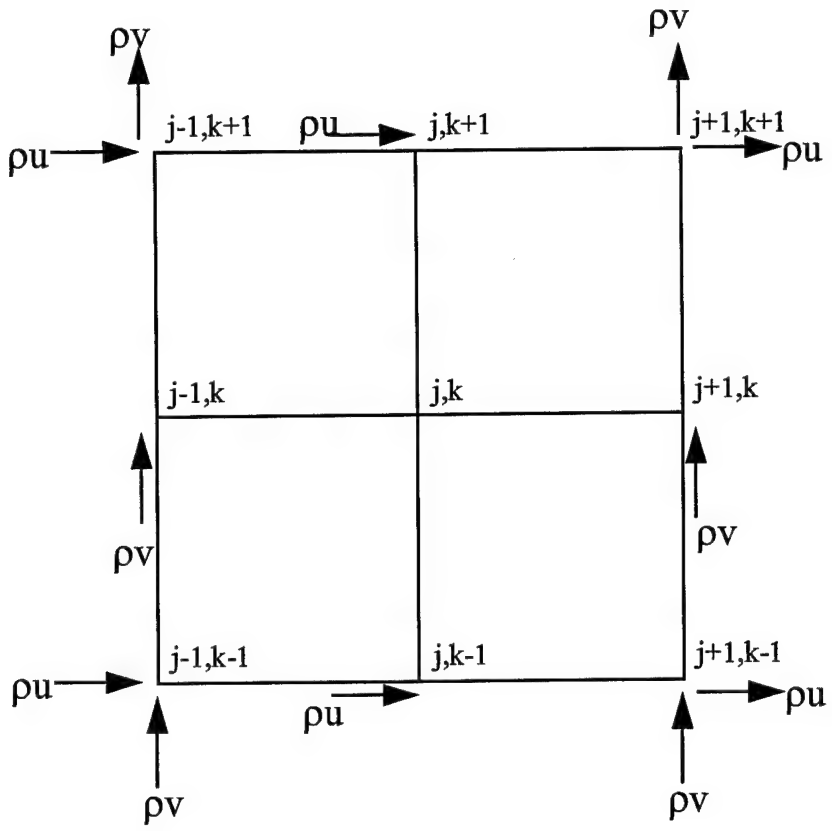

Figure 5 Obstacle boundary conditions

As can be seen in the figure above a 3 x 3 grid is the smallest obstacle possible to provide for the specification of quantities inside an obstacle. The figure above also shows what quantities exist at the points on the boundaries. The quantities that exist at the interior point are density and energy.

For inviscid flow on rigid surfaces four distinct boundary conditions exist. The first is that the component of the x-momentum normal to the surface is zero. Similarly, the component of the y-momentum normal to the surface is also zero. The third condition is that the pressure gradient is zero.

The last boundary condition is that velocity component normal to the obstacle surface is zero.

Another assumption needed to evaluate all parameters is constant energy inside the obstacle, E. The equation for energy is:

$$E = \rho\left(e + \frac{1}{2}\left(u^2 + v^2\right)\right) \tag{79}$$

Since at this point the velocities are known, the energy and the pressure can be calculated. This equation along with the velocities allows for calculating ρ on the surface.

The pressure at the surface of the body obtained by solving the ideal gas law equation, which is:

$$p = \rho RT = (\gamma - 1)\left(E - \frac{1}{2}\rho\left(u^2 + v^2\right)\right) \tag{80}$$

RESULTS OF VARIOUS ATTENUATION STRUCTURES

The following sections detail the results of running the simulation with poles, shear plates and wedges for obstacles.

Before reviewing the results of the simulations it is helpful to examine the flow field results without any obstacles present. All of the results to be presented are at 30 time steps. The program does utilize variable time stepping so the results are not at the same total time. These results are shown in the figure below.

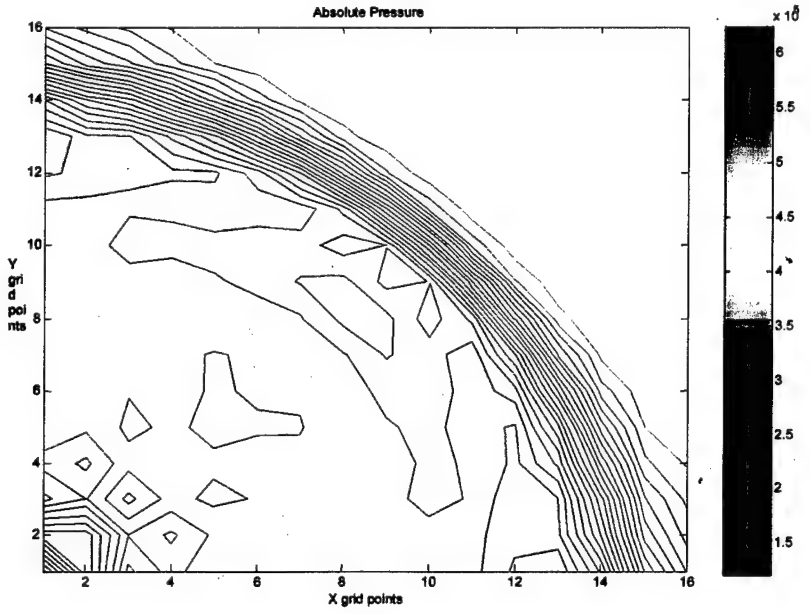

Figure 6 2-D flow field results without obstacles

The figure above illustrates two points unique to numerical solutions. The first point is the gradient at the shockfront. The gradient of the shock is shown as the lines spaced closely together. These lines follow the curved path of the shockwave and travel at the

speed of the shock. The other point that is well illustrated, are the numerical oscillations inherent in this type of solution. The regions outlined behind the shockfront are, in fact, numerical oscillations of the solution and do not exist in the real explosion

POLES

The results for the field of poles were disappointing since it was the conceptual basis for this thesis. The pole obstacles, in the figure below, were set at 3 by 3 grid points with infinite height. The actual physical size of the obstacles varied with the total number of grid points and the calculation distance. For example a 10 by 10 grid, with a 4 meter calculation distance generates a pole that is 1.2 maters on a side. It is very important to ensure that the obstacle size is reasonable. In the above example to make the pole dimensions smaller a greater total number of grid points would be needed or a smaller calculation distance.

Following the completion of many simulations the effect of poles on reducing the pressure of a blast wave it was found that the effect is minimal. The blast wave does compress the air on the blast side of the pole as I expected, but the shadow of the pole does not extend far enough in space to actually cause a decrease in the pressure of the wave after it has passed through the entire field. The cause of this phenomenon is the low viscosity of air. The calculations that are used for the simulation are inviscid, which allows the air to flow easily around a small object such as a pole. For the effect of an obstacle to be maintained farther downrange, the obstacle must be of a size large enough to produce a significant wake or a field of small poles set in a grid or staggered grid pattern that are close together. The field of poles would need to be set on the order of their diameter to be effective. Due to simulations running on a desktop computer system I could not generate a field of poles of sufficient density to demonstrate this directly. I am extrapolating a solution based upon the limited observations and conclusions I made following the simulations. The following figure illustrates this result.

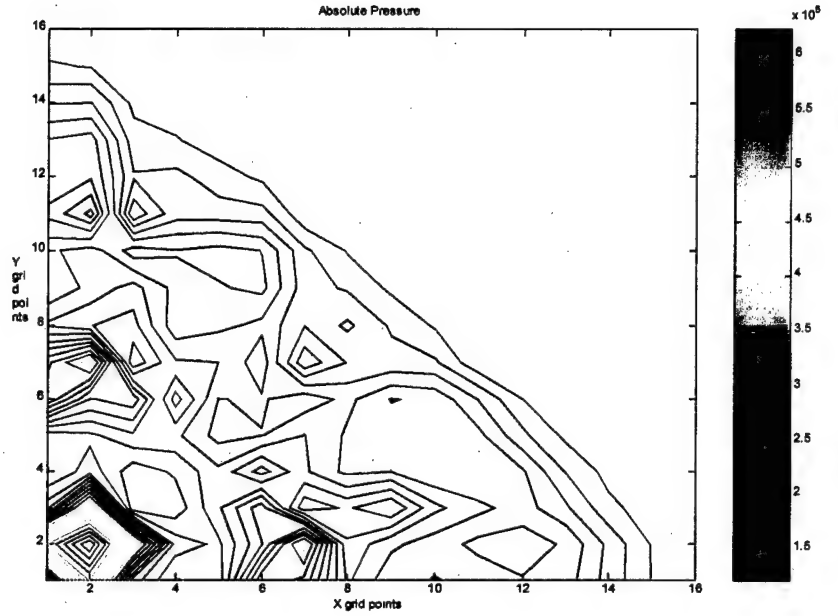

Figure 7 Flow field results for pole obstacles

This figure shows that the first pole, located at grid point (3,3) has a very large pressure gradient on its face. This gradient is produce by the compression of the air between the immovable pole and the moving air behind. This compressed air contains a great deal of energy that remains stagnant upon the face of this pole.

The figure also shows that following passage of the shock through the pole field the gradient has been spread over a larger area, since the contour lines are farther apart, but the magnitude of the gradient has not changed. Therefore, the shock is not as steep as it was originally but is of essentially the same magnitude. Thus, the field of poles has 'softened' the shock but has not attenuated the peak pressure. The interpretation of this result is that this spacing of poles, which is an example of many simulation runs, does not achieve the desired outcome. Extending this interpretation further leads to the conclusion

that no economical pole field would attenuate a shock to a level low enough to protect a structure.

SHEAR PLATES

A shear plate is a thin long plate with the long axis ideally placed parallel to the direction of flow. Due to the uncertainty in the location of an explosive device, I placed the shear plates such that the length axis is perpendicular to the face of the protected building. The shear plates, in the figure below, are 3 grid points wide by 5 grid points long. I performed numerous simulations with other sizes and orientations but this combination of width, length and placement best demonstrates the effect of shear plates on the shock.

The shear plates performed slightly better than the pole field largely due to their size and orientation. The first plate redirects the majority of the flow field in such a way that it no longer raises the pressure on the lower surfaces of subsequent obstacles. The pressure on the upper side of each shear plate is significantly less than the pressure on the lower side. This is the result of the lower plates acting as walls and shielding the upper plates from the blast pressure. The plates also redirect the flow by straightening it in the length direction of the plates. After the flow has passed the end of each plate it begins to diffract. The placement of the shear plates is critical to achieve this effect.

The results of the simulation can be seen in the following figure.

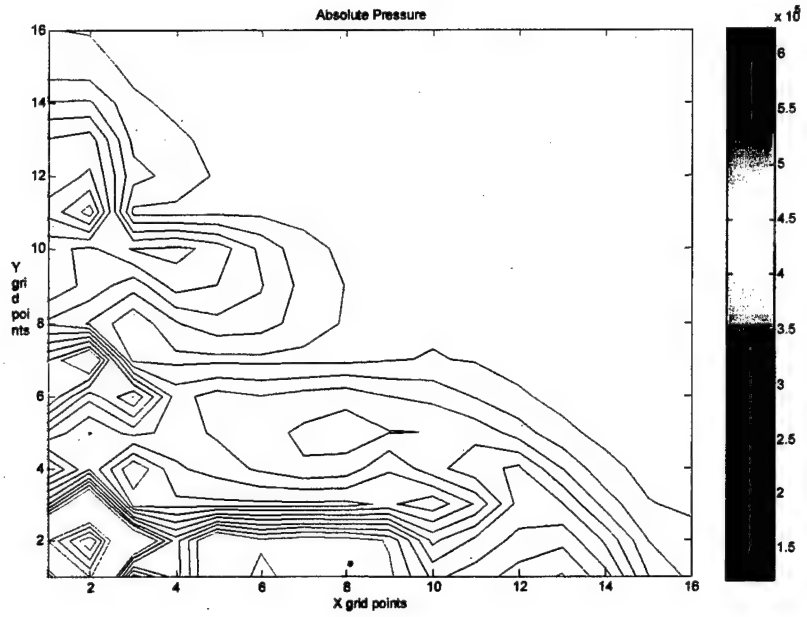

Figure 8 Flow field results for shear plate obstacles

The figure above demonstrates the effect of shear plates on the flow. The lower plate redirects the majority of the flow along the lower edge of the computational realm and removes a great deal of the energy from the flow that impacts the middle and top plates. The flow then diffracts around the lower plate after it has passed. The gradient is again 'softened', but its magnitude remains essentially the same as the original unimpeded shock. The large gradient in the lower left corner is produced by the shock impacting upon the face of the lower shear plate, which is immovable. This impact compresses the air and produces a large pressure peak in a very small area. The middle and top shear plates have similar gradients on their faces but the magnitude is much smaller.

To provide significant protection for a building the location of the explosive would either need to be known or surmised before construction of the plates. The plates could then be

placed in such a way as to direct the flow away from the protected structure. Unfortunately, foresight is rarely this accurate and the construction of many plates (or walls for larger buildings) will be neither aesthetically pleasing nor particularly inexpensive.

WEDGES

A wedge is to two thin plates placed at an angle to each other and joined at the apex. The wedges used in this simulation consist of two plates 3 grid points by 5 grid points placed perpendicular to each other. The wedges were then placed into the computational realm such that the lower face of the wedge is diagonally across the path of the shock. The wedge in this position acts somewhat as a shield or wall in the path of the shock.

The results for the wedge obstacles were similar to the shear plates. A large pressure gradient builds on the explosion side of the lower face and a shadow, or wake forms behind the wedge in the hollow interior. As the flow progresses past the end of the wedge diffraction begins and the shadow disappears. The main difference between the shear plates and the wedge obstacle is that, due to its size and geometry, the wedge does a better job of lowering the pressure in its wake. The wake, however, disappears rapidly, due to diffraction, after the shock passes the end of the obstacle. The orientation of the obstacle is not as critical as with the shear plates. The figure below illustrates these points

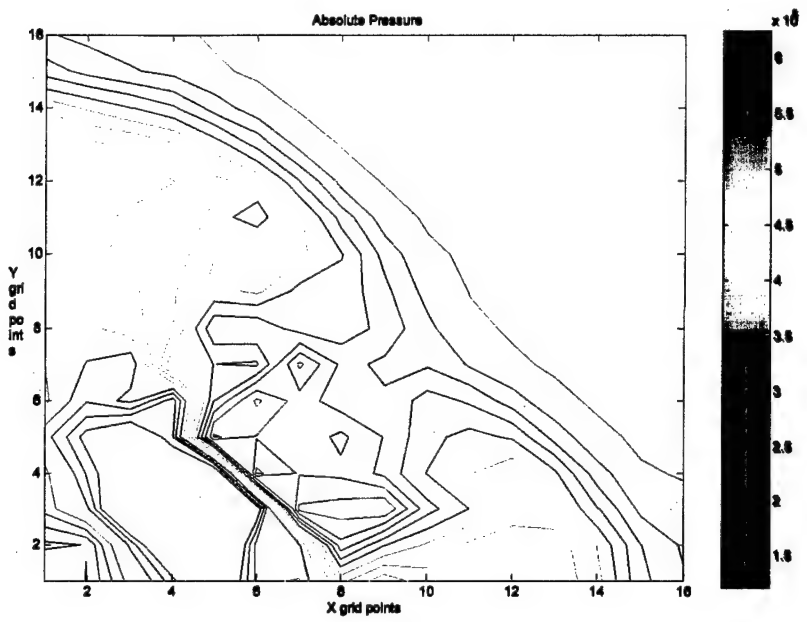

Figure 9 Flow field results for wedge obstacle

The previous figure illustrates the effects of a wedge obstacle on the flow. The geometry of the wedge is such that one face is roughly perpendicular to the shock and a large pressure gradient builds on this face. This is shown by the closely spaced contour lines in the lower left corner between grid (4,4) and (7,2). The effects of this pressure gradient are that, the wedge absorbs a large amount of energy, the flow is split and a shadow or wake forms inside the faces of the wedge. The pressure gradient downstream of the wedge is 'softened', but its magnitude is, once again, similar to the unimpeded shock. Once again, following passage of the shock past the end of the wedge diffraction begins and the pressure begins to fill the shadow. This result is not unexpected.

Since the orientation of the wedge is not as critical to its performance as that of the shear plates, wedges would be more a practical protection than the other obstacles investigated. However, due to the low viscosity of air the wedge would, in all likelihood have to be incorporated as an element on the exterior of the structure, and not a separate element.

This incorporation would lead to a structure with wedge shaped faces which, it is already known to demonstrate good resistance to surface loading from many sources including blasts, waves, running water etc. Finally this form of protection is neither inexpensive nor aesthetically pleasing.

CHAPTER 6: COST ANALYSIS

The cost analyses for the obstacles investigated are moot. Since the pole obstacle does not work as hoped, unless many poles are placed closely together, in which case a wall would most likely be cheaper, I will present a synopsis of the available cost analyses for the conventional approach to blast protection. The following text is a compilation of others work and is not to be considered my own. These are my words but not my ideas or effort.

Nuclear Disasters and the Built Environment contains an overview of the results of a testing completed in the 1950's with above ground nuclear weapons. The results show that an overpressure of 5 psi (34.4 kPa) will completely demolish a conventional wood frame house and an overpressure of 1.7 psi (11.7 kPa) will result in serious damage. A later test series included strengthened wood frame houses. The first house, which was subjected to 4 psi (27.6 kPa) overpressure, sustained serious structural damage but remained standing. The second house was exposed to 2.6 psi (17.9 kPa) and suffered damage similar to the unreinforced house exposed to 1.7 psi. Also included in the second test were two brick houses. These were found to be of the same strength as the wood frame houses. One brick house was exposed to 5 psi (34.4 kPa) and sustained damage similar to the wood frame house. The second house received an overpressure of 1.7 psi (11.7 kPa) and sustained much less damage than the wood frame house.

The strengthening of the houses for the test resulted in a 5% construction premium over unreinforced construction.

The table below compiles the strengths of some common building materials. As can be seen the overpressure required for failure is quite low.

Table 11 Strength of common building materials

Structural element	Overpressure to cause failure (psi)
Unreinforced glass	0.5-1.0
Corrugated steel or aluminum siding	1.0-2.0
Brick wall, unreinforced	3.0-10.0
Standard house construction	1.0-2.0
Concrete wall panels 8"-12" thick	1.5-5.5

Protecting Buildings from Bomb Damage contains an extensive and thorough cost analysis of hardening a new building to resist exterior explosions. I will summarize the results of the analysis. The model was based upon a 5% construction premium for hardening the building and also assumed a minimum 10% return on investment. The construction premium is the additional cost that would be incurred if the blast hardening were to be included in the construction of a building. The construction premium assumption is in keeping with the result from the tests completed in the 1950's. The output from the model was the lease premium based upon these assumptions. These results are shown in the table below.

Table 12 Construction-lease premium

Assume construction premium	Resulting lease premium
3%	2.69%
5%	3.46%
7%	4.23%

The lease premium in the table above is the additional amount that would have to be charged lessees in a blast hardened building. This lease premium would recover the construction premium at the 10% return on investment. The table clearly shows that the resulting lease premium is less than the construction premium. Therefore it can be economically feasible to provide blast protection in new construction, if the developer or customer deems it necessary.

BIBLIOGRAPHY

Roache, Patrick. Computational Fluid Dynamics. Albuquerque, New Mexico. Hermosa Publishers, 1976

Steadman, Phillip and Hodgkinson, Simon. Nuclear Disasters & the Built Environment. London. Butterworth & Co. Ltd., 1990

Shapiro, Ascher. The Dynamics and Thermodynamics of Compressible Fluid Flow. Vol. 2. New York. The Ronald Press Company, 1954

NASA Conference Publication 2201. Numerical Boundary Condition Procedures. NASA Scientific and Technical Information Branch, 1981

Ames, William. Numerical Methods for Partial Differential Equations. New York. Barnes & Noble, Inc, 1969

Richtmeyer, Robert and Morton, K. Difference Methods for Initial Value Problems. 2nd ed. New York. Interscience Publishers, 1967

Davis, Julian. Finite Difference Methods in Dynamics of Continuous Media. New York. MacMillan Publishing Company, 1986

Bollinger, G. Blast Vibration Analysis. Carbondale and Edwardsville, Illinois. Southern Illinois University Press, 1971

National Research Council. Protecting Buildings from Bomb Damage. Washington, D. C. National Academy Press, 1995

Mays, G. and Smith, P. Blast Effects on Buildings. London. Thomas Telford Publications, 1995

Bazhenova, T. et al. Shock Waves in Real Gases. NASA, 1968

Anderson, Dale et al. Computational Fluid Mechanics and Heat Transfer. New York. Hemisphere Publishing Corporation, 1984

Bowen, J. et al. Shock Waves, Explosions, and Detonations. Vol 87. New York. American Institute of Aeronautics and Astronautics, Inc, 1983

Mitchell, A. and Griffiths, D. The Finite Difference Method in Partial Differential Equations. Chichester, England. John Wiley & Sons, 1980

Eberhardt, Scott. Computational Fluid Dynamics I. Course Notes Winter 1997. Seattle, Washington. ASUW Publishing, 1987

APPENDIX A: MACCORMACK 1-D CODE LISTING

```
%-------------------------------------------------------------
%                          Some constants needed for calculations        (UNITS)
%-------------------------------------------------------------
%Courant number
CFL=input('Enter the Courant number(0<CFL<1 0.5 is a good starting point): ');
                if CFL>1.
                error('The Courant number must be less than 1!')
                elseif CFL<=0.
                error('The Courant number must be larger than zero!')
                end
%-------------------------------------------------------------
%gamma for air
g=1.4;
g1=g-1;
%-------------------------------------------------------------
%rho = initial density
                (kg/m^3)
rho=1.614;
%-------------------------------------------------------------
%Pin = initial pressure                                                  (Pa)
Pin=101325;
%-------------------------------------------------------------
%T=initial temperature
                (K)
T1=298;
%-------------------------------------------------------------
%calculate the speed of sound at the initial pressure
                                                        (m/s)
c=(1.4*Pin/rho)^(1/2);
%-------------------------------------------------------------
%h= distance over which to perform calculations          (m)
h=input ('Enter the distance in meters over which to perform calculations: ');
                if h<=0.
                error('You must enter a nonzero distance!')
                end
%-------------------------------------------------------------
%stop= number of position steps per distance
stop=input ('Enter the total number of x grid points(101 is a good starting point): ');
                if stop<=0.
```

```
                error('You must enter a nonzero number of x grid points!')
                end;
%-------------------------------------------------------
%calculate the x step size
dx=h/(stop-1);
%-------------------------------------------------------
%                                        Print output of values on screen
%-------------------------------------------------------
fprintf('-------------------------------------------------------------\n')
fprintf('                                        Calculation values
                \n')
fprintf('The Courant number is.....%1.2f\n',CFL)
fprintf('The distance to calculate over is.....%3.2f m\n',h)
fprintf('The number of x grid points is.....%4.0f\n',stop-1)
fprintf('The x step size is.....%1.4f m\n',dx)
fprintf('-------------------------------------------------------------\n')
fprintf('**************************************************************
*\n')
fprintf('                        Conditions ahead of the shock
\n')

fprintf('Gamma is.....%1.1f\n',g)
fprintf('The initial pressure is.....%3.3g Pa\n',Pin)
fprintf('The initial density is.....%1.4f kg/m^3\n',rho)
fprintf('The initial temperature is.....%3.3g K\n',T1)
fprintf('The speed of sound is.....%3.4f m/s\n',c)
fprintf('**************************************************************
*\n')
%-------------------------------------------------------
endt=input('Enter the number of time steps to calculate over: ');
                if endt<=0
                error('You must enter a number of time steps larger than zero!')
                end
%-------------------------------------------------------
%define vector size

oldp=[1:stop]*0.0;
newp=[1:stop]*0.0;
oldr=[1:stop]*0.0;
newr=[1:stop]*0.0;
barr=[1:stop]*0.0;
oldru=[1:stop]*0.0;
newru=[1:stop]*0.0;
```

```
barru=[1:stop]*0.0;
newu=[1:stop]*0.0;
oldE=[1:stop]*0.0;
newE=[1:stop]*0.0;
barE=[1:stop]*0.0;
loc=[1:stop]*0.0;
%-------------------------------------------------------
%               Convert x grid points to actual distances

for x=1:stop
                loc(x)=(x-1)*dx;
end
%-------------------------------------------------------
%               Calculate Peak staic overpressure Ps based on charge size
%                and range to charge the driving force for the
%                calculations to follow
%-------------------------------------------------------
%                                       r=range to charge
                (m)
r=input('Enter the range to the charge in meters: ');
                if r<=0
                error('You must enter a range larger than zero!')
                end
%-------------------------------------------------------
%               Print out table of explosive types
%                                       T=type of explosive
fprintf('-------------------------------------------\n')
fprintf('       Choose the explosive using the table below\t\n')
fprintf('Compound B .....1\n')
fprintf('RDX (Cyclonite).....2\n')
fprintf('HMX .....3\n')
fprintf('Nitroglycerin (liquid).....4\n')
fprintf('TNT .....5\n')
fprintf('Blasting Gelatin .....6\n')
fprintf('60 percent Nitroglycerin dynamite .....7\n')
fprintf('Semtex .....8\n')
fprintf('-------------------------------------------\n')
%-------------------------------------------------------
%               Ask for whiat type of explosive to use
%-------------------------------------------------------
T=input('enter the type of explosive to use: ');
                if T==1
                                        S=1.148;
```

```
                elseif T==2
                                S=1.185;
                elseif T==3
                                S=1.256;
                elseif T==4
                                S=1.481;
                elseif T==5
                                S=1.000;
                elseif T==6
                                S=1.000;
                elseif T==7
                                S=0.600;
                elseif T==8
                                S=1.250;
                else
                error('You must enter an explosive type!')
                end
fprintf('The TNT conversion factor for this explosive is: %1.3f\n',S)
%-------------------------------------------------------
%               Ask what charge mass to use                     (kg of TNT)
%-------------------------------------------------------
M=input('Enter the mass of the charge in kilograms: ');
                if M<0
                error('You must enter a mass larger than zero!')
                end
W=M*S;
%-------------------------------------------------------
%                               calculate peak static overpressure (Bar)

z=r/(W^(1/3));
Ps=(6.7/(z^3));
                if Ps<10
                                Ps=((0.975/z)+(1.455/(z^2))+(5.85/(z^3)))-0.019;
                else Ps=(6.7/(z^3));
                end

                if Ps<0.1
                                Ps=Ps;
                end

%-------------------------------------------------------
%               Convert Ps(the static overpressure) from Bar (Pa)
%                to Pascals from overpressure to absolute pressure
```

```
%              and apply conversion factor for ground burst
%-------------------------------------------------------
Ps=1.8*((Ps*100000)+Pin);


%-------------------------------------------------------
%                                        Print output of values on screen
fprintf('-------------------------------------------------------\n')
fprintf('                                        Explosive values
            \n')
fprintf('The range to the charge from the x=0 point is.....%4.2f m\n',r)
fprintf('The mass of the charge in TNT equivalents is.....%4.3f kg\n',W)
fprintf('The static overpressure of the charge is.....%1.3g Pa\n',Ps-Pin)
fprintf('-------------------------------------------------------\n')
%-------------------------------------------------------
%                                        Calculate intial velocity using Ps
%-------------------------------------------------------
Mach=sqrt(((Ps/Pin)-1)*((g+1)/(2*g))+1);
Vin=c*(2/(g+1)*(Mach-(1/Mach)));
%                                        Print output of values on screen
fprintf('The initial wave velocity is.....%4.4f m/s\n',Vin)
fprintf('The shock wave Mach number is.....%2.4f\n',Mach)
fprintf('-------------------------------------------------------\n')
%-------------------------------------------------------
%                                        Calculate values behind the shock
%-------------------------------------------------------
rho2=rho/(1-(2/(g+1))*(1-(1/Mach^2)));
T2=Ps/(rho2*(8314.51/29));
c2=(c^2*T2/T1)^0.5;
fprintf('**********************************************************************
*\n')
fprintf('                        Conditions behind the shock
\n')
fprintf('The pressure is.....%1.3g Pa\n',Ps)
fprintf('The density is.....%1.4f kg/m^3\n',rho2)
fprintf('The temperature is.....%3.3g K\n',T2)
fprintf('The speed of sound is.....%3.4f m/s\n',c2)
fprintf('**********************************************************************
*\n')

%-------------------------------------------------------
%                                        Define initial conditions
%-------------------------------------------------------
```

```
for             x=1:stop
                oldp(x)=Pin;
                oldr(x)=rho;
                oldru(x)=0;
                oldE(x)=Pin/g1;
                end
%---------------------------------------------------------
%               Define boundary conditions at grid point x=1
%---------------------------------------------------------
                oldp(1)=Ps;
                oldr(1)=rho/(1-(2/(g+1))*(1-(1/Mach^2)));
                oldru(1)=oldr(1)*Vin;
                oldE(1)=(Ps/g1)+(0.5*oldru(1)^2/oldr(1));


%---------------------------------------------------------
%               Calculate the new speed of sound based on new pressure
%                and density values
%---------------------------------------------------------
                newc=sqrt(1.4*oldp(x)/oldr(x));
%---------------------------------------------------------
%                                       Calculate the initial time step
%---------------------------------------------------------
    dt=CFL*dx/(newc+oldru(1)/oldr(1));
fprintf('The inital time step is.....%1.5e s\n',dt)
%---------------------------------------------------------
%                                       Initialize total time counter
%---------------------------------------------------------
tt=0;
%---------------------------------------------------------
%               Define predictor values at x equal 1
%---------------------------------------------------------
                barr(1)=oldr(1)-(dt)*(oldru(2)-oldru(1));
                barru(1)=oldru(1)-(dt)*(((oldru(2)^2/oldr(2))-...
                  (oldru(1)^2/oldr(1)))+(oldp(2)-oldp(1)));
                barE(1)=oldE(1)-(dt)*((oldE(2)+oldp(2))*oldru(2)/...
                  oldr(2)-(oldE(1)+oldp(1))*oldru(1)/oldr(1) );

%---------------------------------------------------------
%               Adjust x grid end point for finite differencing

stop=stop-1;
%---------------------------------------------------------
```

```
%               Finite difference for density, momentum and energy
%-------------------------------------------------------------


for t=1:endt
                dtmin=1.;
                CFLmin=CFL;
                dtdx=dt/(dx);
%Calculate total time
                tt=tt+dt;
                for x=2:stop

%-------------------------------------------------------------
%                                                       PREDICTOR
%-------------------------------------------------------------
%               Solve as a function of time and position for increasing
%                time
%-------------------------------------------------------------
%               Density(r)

                barr(x)=oldr(x)-(dtdx)*(oldru(x+1)-oldru(x));
%-------------------------------------------------------------
%               Momentum(ru)

                barru(x)=oldru(x)-(dtdx)*((oldru(x+1)^2/oldr(x+1))-...
                  (oldru(x)^2/oldr(x))+(oldp(x+1)-oldp(x)));
%-------------------------------------------------------------
%               Energy(E)

                barE(x)=oldE(x)-(dtdx)*((oldE(x+1)+oldp(x+1))*...
                  (oldru(x+1)/oldr(x+1))-(oldE(x)+oldp(x))*...
                  (oldru(x)/oldr(x)));
%-------------------------------------------------------------
%                                                       CORRECTOR
%-------------------------------------------------------------
%               Density(r)

                newr(x)=(barr(x)+oldr(x))/2-(dtdx/2)*(barru(x)-barru(x-1));

%               Left wall boundary conditions

                newr(stop+1)=newr(stop-1);
                newr(stop)=newr(stop-1);
```

```
%-------------------------------------------------------
%               Momentum(ru)

                newru(x)=(barru(x)+oldru(x))/2-(dtdx/2)*((barru(x)^2/...
                  barr(x))-(barru(x-1)^2/barr(x-1))+(g1*(barE(x)-...
                  (.5*(barru(x)^2/barr(x)))))-(g1*(barE(x-1)-(.5*...
                  (barru(x-1)^2/barr(x-1))))));

%               Left wall boundary conditions

                newru(stop)=0;
                newru(stop+1)=0;

%-------------------------------------------------------
%               Energy(e)

                newE(x)=(barE(x)+oldE(x))/2-(dtdx/2)*((barru(x)/...
                  barr(x))*(barE(x)+(g1*(barE(x)-(.5*(barru(x)^2/...
                  barr(x))))))-((barru(x-1)/barr(x-1))*(barE(x-1)+...
                  (g1*(barE(x-1)-(.5*(barru(x-1)^2/barr(x-1)))))));

%-------------------------------------------------------
%               Calculate pressure using the continuity equation for the
%                ideal gas law and the new values for energy, momentum
%                and density

                newp(x)=g1*(newE(x)-(.5*(newru(x)^2/newr(x))));

                newp(stop+1)=newp(stop-1);


%-------------------------------------------------------
%               Calculate velocity using the new values for momentum
%                and density

                newu(x)=newru(x)/newr(x);
%-------------------------------------------------------
%                                                  Check stability
%                                       Calculate the new speed of sound

                newc=sqrt(g*newp(x)/newr(x));
```

```
%-------------------------------------------------------
%                                   Calculate the new time step dt

            dtnew=dx/(newu(x)+newc);
            if dtnew < dtmin
                        dtmin=dtnew;
            end
            CFLnew=(dtmin/dx)*((newc+oldru(x)/oldr(x)));
            if CFLnew<CFLmin
                        CFLmin=CFLnew;
            end
%-------------------------------------------------------
%           Return for next x
            end

%-------------------------------------------------------
%           Apply numerical boundary scheme to calculate all values
%            at x=1 and x=stop+1

            newr(1)=oldr(1);
            newE(1)=oldE(1);
            newru(1)=oldru(1);
            newu(1)=oldru(1)/oldr(1);
            newp(1)=oldp(1);
            barr(1)=oldr(1)-(dt)*(oldru(2)-oldru(1));
            barru(1)=oldru(1)-(dt)*((oldru(2)^2/oldr(2)-...
              oldru(1)^2/oldr(1))+(oldp(2)-oldp(2)));
            barE(1)=oldE(1)-(dt)*((oldE(2)+oldp(2))*oldru(2)/...
              oldr(2) -(oldE(1)+oldp(1))*oldru(1)/oldr(1) );
            newE(stop+1)=oldE(stop+1);
            newu(stop+1)=newru(stop+1)/newr(stop+1);


%-------------------------------------------------------
%           Update time step and CFL(Courant number)

    dt=dtmin;
            CFL=CFLmin;
fprintf('CFL= %1.3g\tdelta t= %1.5e s\tTotal time= %1.5e s\n',CFL,dt,tt)


%-------------------------------------------------------
```

```
%               Output the values for pressure, density and velocity
%                using graphs
%-------------------------------------------------------------
%               Pressure
subplot(3,1,1), plot(loc,newp), ylabel('Absolute Pressure (Pa)')
%-------------------------------------------------------------
%               Density
subplot(3,1,2), plot(loc,newr), ylabel('Density (kg/m^3)')
%-------------------------------------------------------------
%               Velocity

subplot(3,1,3), plot(loc,newu), ylabel('Velocity (m/s)'),...
                xlabel('Distance (m)')

%-------------------------------------------------------------
pause(.0001)
%-------------------------------------------------------------
%Replace old values with new values and begin next time step

oldp=newp;
oldr=newr;
oldru=newru;
oldE=newE;
%-------------------------------------------------------------
%               Return for next t
end
%-------------------------------------------------------------
```

APPENDIX B: MACCORMACK QUASI 2-D CODE LISTING

```
clear all
%-----------------------------------------------------------

%                          Some constants needed for calculations (UNITS)

%Courant number
CFL=input('Enter the Courant number(0<CFL<1 0.5 is a good starting point): ');
                if CFL>1.
                error('The Courant number must be less than 1!')
                elseif CFL<=0.
                error('The courant number must be larger than zero!')
                end
%-----------------------------------------------------------
%gamma for air
g=1.4;
g1=g-1;

%-----------------------------------------------------------
%rho = initial density
                (kg/m^3)
rho=1.614;

%-----------------------------------------------------------
%Pin = initial pressure                                                  (Pa)
Pin=101325;

%-----------------------------------------------------------
%T=initial temperature
                (K)
T1=298;

%-----------------------------------------------------------
%calculate the speed of sound at the initial pressure          (m/s)
c=(1.4*Pin/rho)^(1/2);

%-----------------------------------------------------------
%h= distance over which to perform calculations                          (m)
h=input ('Enter the distance in meters over which to perform calculations: ');
                if h<=0.
                error('You must enter a nonzero distance!')
                end
```

```
%-----------------------------------------------------------
%stop= number of position steps per distance
stop=input ('Enter the total number of grid points in the x,y directions(31 is a good
starting point): ');
                if stop<=0.
                error('You must enter a nonzero number of x grid points!')
                end;

%-----------------------------------------------------------
%calculate the x and y step size
dx=h/(stop-1);
dy=h/(stop-1);

%-----------------------------------------------------------

%                                          Print output of values on screen
fprintf('-------------------------------------------------\n')
fprintf('                                    Calculation values
                \n')
fprintf('The Courant number is.....%1.2f\n',CFL)
fprintf('The x distance to calculate over is.....%3.2f m\n',h)
fprintf('The y distance to calculate over is.....%3.2f m\n',h)
fprintf('The number of x grid points is.....%4.0f\n',stop-1)
fprintf('The number of y grid points is.....%4.0f\n',stop-1)
fprintf('The x step size is.....%1.4f m\n',dx)
fprintf('The y step size is.....%1.4f m\n',dx)
fprintf('-------------------------------------------------------\n')
fprintf('*************************************************************
*\n')
fprintf('                        Conditions ahead of the shock
\n')

fprintf('Gamma is.....%1.1f\n',g)
fprintf('The initial pressure is.....%3.3g Pa\n',Pin)
fprintf('The initial density is.....%1.4f kg/m^3\n',rho)
fprintf('The initial temperature is.....%3.3g K\n',T1)
fprintf('The speed of sound is.....%3.4f m/s\n',c)
fprintf('*************************************************************
*\n')

%-----------------------------------------------------------

endt=input('Enter the number of time steps to calculate over: ');
```

```
                if endt<=0
                error('You must enter a number of time steps larger than zero!')
                end

%-------------------------------------------------------
%                                       define vector size

for             x=1:stop
for             y=1:stop
op(x,y)=0.0;
np(x,y)=0.0;
Olr(x,y)=0.0;
nr(x,y)=0.0;
br(x,y)=0.0;
Olru(x,y)=0.0;
nru(x,y)=0.0;
Olrv(x,y)=0.0;
nrv(x,y)=0.0;
bru(x,y)=0.0;
nu(x,y)=0.0;
brv(x,y)=0.0;
nv(x,y)=0.0;
oE(x,y)=0.0;
nE(x,y)=0.0;
bE(x,y)=0.0;


end
end

%-------------------------------------------------------
%               Calculate Peak static overpressure Ps based on charge
%                size and range to charge the driving force for the
%                calculations to follow
%-------------------------------------------------------
%                                   r=range to charge
                (m)
%-------------------------------------------------------

r=input('Enter the range to the charge in meters: ');
                if r<=0
                error('You must enter a range larger than zero!')
                end
```

```
%               Print out table of explosive types
%                               T=type of explosive
fprintf('-------------------------------------------------\n')
fprintf('       Choose the explosive using the table below\t\n')
fprintf('Compound B .....1\n')
fprintf('RDX (Cyclonite).....2\n')
fprintf('HMX .....3\n')
fprintf('Nitroglycerin (liquid).....4\n')
fprintf('TNT .....5\n')
fprintf('Blasting Gelatin .....6\n')
fprintf('60 percent Nitroglycerin dynamite .....7\n')
fprintf('Semtex .....8\n')
fprintf('-------------------------------------------------\n')

%---------------------------------------------------------
%               Ask for what type of explosive to use
%---------------------------------------------------------

T=input('enter the type of explosive to use: ');
                if T==1
                                S=1.148;
                elseif T==2
                                S=1.185;
                elseif T==3
                                S=1.256;
                elseif T==4
                                S=1.481;
                elseif T==5
                                S=1.000;
                elseif T==6
                                S=1.000;
                elseif T==7
                                S=0.600;
                elseif T==8
                                S=1.250;
                else
                error('You must enter an explosive type!')
                end
fprintf('The TNT conversion factor for this explosive is: %1.3f\n',S)

%---------------------------------------------------------
```

```
%               Ask for mass of charge
                (kg)
%---------------------------------------------------------

M=input('Enter the mass of the charge in kilograms: ');
                if M<0
                error('You must enter a mass larger than zero!')
                end
W=M*S;

%---------------------------------------------------------
%                               Calculate peak static overpressure          (B)
%---------------------------------------------------------

z=r/(W^(1/3));
Ps=(6.7/(z^3));
                if Ps<10
                                Ps=((0.975/z)+(1.455/(z^2))+(5.85/(z^3)))-0.019;
                else Ps=(6.7/(z^3));
                end

                if Ps<0.1
                                Ps=Ps;
                end

%---------------------------------------------------------
%               Convert Ps(the static overpressure) from Bar (Pa)
%                to Pascals from overpressure to absolute
%                pressure and apply conversion factor for
%                ground burst
%---------------------------------------------------------

Ps=1.8*((Ps*100000)+Pin);


%---------------------------------------------------------
%                                               Print output of values on screen
%---------------------------------------------------------

fprintf('-------------------------------------------------------\n')
fprintf('                                       Explosive values
                \n')
fprintf('The range to the charge from the x=0 point is.....%4.2f m\n',r)
```

```
fprintf('The mass of the charge in TNT equivalents is.....%4.3f kg\n',W)
fprintf('The static overpressure of the charge is.....%1.3g Pa\n',Ps-Pin)
fprintf('-----------------------------------------------------\n')

%-------------------------------------------------------
%                                   Calculate intial velocity using Ps
%-------------------------------------------------------

Mach=sqrt(((Ps/Pin)-1)*((g+1)/(2*g))+1);
Vin=c*(2/(g+1)*(Mach-(1/Mach)));
%                                   Print output of values on screen
fprintf('The initial wave velocity is.....%4.4f m/s\n',Vin)
fprintf('The shock wave Mach number is.....%2.4f\n',Mach)
fprintf('-----------------------------------------------------\n')

%-------------------------------------------------------
%                                   Calculate values behind the shock
%-------------------------------------------------------

rho2=rho/(1-(2/(g+1))*(1-(1/Mach^2)));
T2=Ps/(rho2*(8314.51/29));
c2=(c^2*T2/T1)^0.5;
fprintf('*************************************************************
*\n')
fprintf('                    Conditions behind the shock
\n')
fprintf('The pressure is.....%1.3g Pa\n',Ps)
fprintf('The density is.....%1.4f kg/m^3\n',rho2)
fprintf('The temperature is.....%3.3g K\n',T2)
fprintf('The speed of sound is.....%3.4f m/s\n',c2)
fprintf('*************************************************************
*\n')
%-------------------------------------------------------

%-------------------------------------------------------
%                                   Define initial conditions
%-------------------------------------------------------

for         x=1:stop
for         y=1:stop
            op(x,y)=Pin;
            Olr(x,y)=rho;
            Olru(x,y)=0;
```

```
                Olrv(x,y)=0;
                oE(x,y)=Pin/g1;
end

end

%-------------------------------------------------------
%               Define boundary conditions at grid point x=1 y=1
%-------------------------------------------------------
                y=1;
for             x=1:stop
                op(x,y)=Ps;
                Olr(x,y)=rho/(1-(2/(g+1))*(1-(1/Mach^2)));
                Olru(x,y)=0;
                Olrv(x,y)=Olr(x,y)*Vin;
                oE(x,y)=(Ps/g1)+(0.5*(Olru(x,y)^2+Olrv(x,y)^2)/...
                  Olr(x,y));

end


%-------------------------------------------------------
%               Calculate the new speed of sound based on new pressure
%                and density values
%-------------------------------------------------------

                nc=sqrt(g*op(x,y)/Olr(x,y));
%-------------------------------------------------------
%                                       Calculate the initial time step
%-------------------------------------------------------

    dt=CFL*(1/(abs(Olru(1,1)/Olr(1,1))/dx+abs(Olrv(1,1)/Olr(1,1))/...
                dy+nc*(1/dx^2+1/dy^2)^(0.5)));



fprintf('The inital time step is.....%1.5e s\n',dt)
%-------------------------------------------------------
%                                       Initialize total time counter
%-------------------------------------------------------

tt=0;
```

```
%-------------------------------------------------------
%               Finite difference for density, momentum and energy
%-------------------------------------------------------

%-------------------------------------------------------
%               Adjust x and y grid end point for finite differencing
%-------------------------------------------------------

stop=stop-1;



for t=1:endt

%-------------------------------------------------------
%               NBS for column 1 and row 1
%-------------------------------------------------------

               dtmin=1.;

               dtdx=dt/dx;
               dtdy=dt/dy;

%-------------------------------------------------------
%                                       Calculate total time
%-------------------------------------------------------

               tt=tt+dt;


%-------------------------------------------------------
               for x=1:stop
               for y=1:stop

%-------------------------------------------------------


%                                                       PREDICTOR
%-------------------------------------------------------
%               Solve as a function of time and position for
%                increasing time
```

```
%-------------------------------------------------------
%                                                Density(r)
%-------------------------------------------------------

            br(x,y)=Olr(x,y)-...
   (dt*(((Olru(x+1,y)-Olru(x,y))/dx)+...
               ((Olrv(x,y+1)-Olrv(x,y))/dy)));

            br(x,stop+1)=br(x,stop);
            br(stop+1,y)=br(stop,y);
            br(stop+1,stop+1)=br(stop,stop);
%-------------------------------------------------------
%                                             Momentum(ru)
%-------------------------------------------------------

            bru(x,y)=Olru(x,y)-dt*...
              ((1/dx*...
              (Olru(x+1,y)^2/Olr(x+1,y)-...
               Olru(x,y)^2  /Olr(x,y)  +...
              (op(x+1,y)-op(x,y)      )))+...
              (1/dy*...
              (Olrv(x,y+1)*Olru(x,y+1)/Olr(x,y+1)-...
               Olrv(x,y)  *Olru(x,y)  /Olr(x,y)  )));

            bru(x,stop+1)=bru(x,stop);
            bru(stop+1,y)=bru(stop,y);
            bru(stop+1,stop+1)=bru(stop,stop);

%-------------------------------------------------------
%                                             Momentum(rv)
%-------------------------------------------------------

            brv(x,y)=Olrv(x,y)-dt*...
              ((1/dx*...
              (Olru(x+1,y)*Olrv(x+1,y)/Olr(x+1,y)-...
               Olru(x,y)  *Olrv(x,y)  /Olr(x,y)  ))+...
              (1/dy*...
              (Olrv(x,y+1)^2/Olr(x,y+1)-...
               Olrv(x,y)^2  /Olr(x,y)+...
              (op(x,y+1)-op(x,y)     ))));

            brv(stop+1,y)=brv(stop,y);
            brv(x,stop+1)=brv(x,stop);
```

```
            brv(stop+1,stop+1)=brv(stop,stop);
%-------------------------------------------------------
%                                                Energy(e)
%-------------------------------------------------------

            bE(x,y)=oE(x,y)-dt*...
              ((1/dx*...
              ((oE(x+1,y)+op(x+1,y))*Olru(x+1,y)/Olr(x+1,y)  -...
               (oE(x,y)+op(x,y)   )*Olru(x,y)  /Olr(x,y)   )+...
                1/dy*...
              ((oE(x,y+1)+op(x,y+1))*Olrv(x,y+1)/Olr(x,y+1)  -...
               (oE(x,y)+op(x,y)   )*Olrv(x ,y)  /Olr(x,y)   )));

            bE(stop+1,y)=bE(stop,y);
            bE(x,stop+1)=bE(x,stop);
            bE(stop+1,stop+1)=bE(stop,stop);
%-------------------------------------------------------

% return for next x
end
% return for next y
end

%-------------------------------------------------------
            nr(:,1)=Olr(:,1);
            nru(:,1)=Olru(:,1);
            nrv(:,1)=Olrv(:,1);
            nE(:,1)=oE(:,1);

for         x=2:stop+1
for         y=2:stop+1

%-------------------------------------------------------
%                                                CORRECTOR
%-------------------------------------------------------
%                                                Density(r)
%-------------------------------------------------------

            nr(x,y)=(br(x,y)+Olr(x,y))/2-...
              ( dtdx/2*(bru(x,y)-bru(x-1,y))+...
                dtdy/2*(brv(x,y)-brv(x,y-1)));
```

```
                nr(1,y)=nr(2,y);

%               Left wall boundary conditions

%-------------------------------------------------------

%                                                   Momentum(ru)
%-------------------------------------------------------
Px=g1*(bE(x,y)-0.5*((bru(x,y)^2/br(x,y))+(brv(x,y)^2/br(x,y))));
Pxm=g1*(bE(x-1,y)-0.5*((bru(x-1,y)^2/br(x-1,y))+(brv(x-1,y)^2/br(x-1,y))));

                nru(x,y)=(bru(x,y)+Olru(x,y))/2-...
                  ((dtdx/2*...
                  (( bru(x,y)  *bru(x,y)  /br(x,y)  +Px )-...
                   ( bru(x-1,y)*bru(x-1,y)/br(x-1,y)+Pxm)))+...
                  (dtdy/2*...
                  (  bru(x,y)  *brv(x,y)  /br(x,y)-...
                     bru(x,y-1)*brv(x,y-1)/br(x,y-1)    ))) ;

%-------------------------------------------------------


                nru(1,y)=nru(2,y);

%                                                   Momentum(rv)
%-------------------------------------------------------

Py=g1*(bE(x,y)-0.5*((bru(x,y)^2/br(x,y))+(brv(x,y)^2/br(x,y))));
Pym=g1*(bE(x,y-1)-0.5*((bru(x,y-1)^2/br(x,y-1))+(brv(x,y-1)^2/br(x,y-1))));

                nrv(x,y)=(brv(x,y)+Olrv(x,y))/2-...
                  ((dtdx/2*...
                  ( bru(x,y)  *brv(x,y)  /br(x,y)    -...
                    bru(x-1,y)*brv(x-1,y)/br(x-1,y) ))+...
                  (dtdy/2*...
                  ( (brv(x,y)*brv(x,y)/br(x,y)+    Py  )-...
                    (brv(x,y-1)*brv(x,y-1)/br(x,y-1)+Pym))));

%-------------------------------------------------------


                nrv(1,y)=nrv(2,y);
```

```
%              Left wall boundary conditions

%--------------------------------------------------------

%                                                    Energy(E)
%--------------------------------------------------------

              nE(x,y)=(bE(x,y)+oE(x,y))/2-...
                ((dtdx/2*...
                ((bru(x,y) /br(x,y) *(bE(x,y) +Px )) -...
                (bru(x-1,y)/br(x-1,y)*(bE(x-1,y)+Pxm)))) +...
                (dtdy/2*...
                ((brv(x,y) /br(x,y) *(bE(x,y)+Py )) -...
                (brv(x,y-1)/br(x,y-1)*(bE(x,y-1)+Pym)))) );
%--------------------------------------------------------
%--------------------------------------------------------
%              Right wall boundary conditions
%--------------------------------------------------------

              nr(x,stop+1)=nr(x,stop-1);
%             nr(x,stop)=nr(x,stop-1);
              nrv(x,stop)=0;
              nrv(x,stop+1)=0;
              nE(x,stop+1)=oE(x,stop+1);

              nE(1,y)=nE(2,y);
%--------------------------------------------------------
%return for next x
              end
%--------------------------------------------------------
%return for next y
              end
%--------------------------------------------------------

for           y=1:stop+1
for           x=1:stop+1
%--------------------------------------------------------
%             Calculate pressure using the continuity equation for
%              the ideal gas law and the new values for energy,
%              momentum and density
%--------------------------------------------------------
```

```
np(x,y)=(g1)*(nE(x,y)-(.5*((nru(x,y)^2/...
  nr(x,y))+(nrv(x,y)^2/nr(x,y)))));

np(x,stop+1)=np(x,stop-1);


%-------------------------------------------------------
%          Calculate velocity using the new values for momentum
%           and density
%-------------------------------------------------------

nu(x,y)=nru(x,y)/nr(x,y);
nv(x,y)=nrv(x,y)/nr(x,y);


%-------------------------------------------------------
%                                                  Check stability
%                                    Calculate the new speed of sound
%-------------------------------------------------------

nc=sqrt(g*np(x,y)/nr(x,y));
%-------------------------------------------------------
%
%                                    Calculate the new time step dt
%-------------------------------------------------------


dtn=CFL*(1/(abs(nu(x,y))/dx+abs(nv(x,y))/dy+...
   nc*(1/dx^2+1/dy^2)^(0.5)));
if dtn < dtmin
                dtmin=dtn;
% end dtmin if statement
end
%-------------------------------------------------------

%-------------------------------------------------------

%-------------------------------------------------------
%return for next x
end
%-------------------------------------------------------
%return for next y
end
```

```
%-----------------------------------------------------------


%-----------------------------------------------------------
%               Update time step and CFL(Courant number)
%-----------------------------------------------------------

    dt=dtmin;

fprintf('delta t= %1.5e s\tTotal time= %1.5e s\n',dt,tt)



%-----------------------------------------------------------
%               Output the values for pressure, density and velocity
%                using graphs
%-----------------------------------------------------------

%               Pressure


colormap(cool)
surf(np)
hold on
pcolor((np)-Pin)
title('Absolute Pressure')
xlabel('X grid points')
ylabel('Y grid points')
zlabel('Pascals')

pause(.01)
hold off
%-----------------------------------------------------------
%               Replace old values with new values and begin
%                next time step
%-----------------------------------------------------------

op=np;
Olr=nr;
Olru=nru;
Olrv=nrv;
oE=nE;
%-----------------------------------------------------------
```

```
%              Return for next t
end
%-------------------------------------------------------------
```

APPENDIX C: MACCORMACK 2-D CODE LISTING

```
clear all
%-------------------------------------------------------------

%                          Some constants needed for calculations (UNITS)

%Courant number
CFL=input('Enter the Courant number(0<CFL<1 0.5 is a good starting point): ');
                if CFL>1.
                error('The Courant number must be less than 1!')
                elseif CFL<=0.
                error('The courant number must be larger than zero!')
                end
%-------------------------------------------------------------
%gamma for air
g=1.4;
g1=g-1;

%-------------------------------------------------------------
%rho = initial density
                (kg/m^3)
rho=1.614;

%-------------------------------------------------------------
%Pin = initial pressure                                                  (Pa)
Pin=101325;

%-------------------------------------------------------------
%T=initial temperature
                (K)
T1=298;

%-------------------------------------------------------------
%calculate the speed of sound at the initial pressure          (m/s)
c=(1.4*Pin/rho)^(1/2);

%-------------------------------------------------------------
%h= distance over which to perform calculations                          (m)
h=input ('Enter the distance in meters over which to perform calculations: ');
                if h<=0.
                error('You must enter a nonzero distance!')
                end
```

```
%-----------------------------------------------------
%stop= number of position steps per distance
stop=input ('Enter the total number of grid points in the x,y directions(31 is a good
starting point): ');
                if stop<=0.
                error('You must enter a nonzero number of x grid points!')
                end;

%-----------------------------------------------------
%calculate the x and y step size
dx=h/(stop-1);
dy=h/(stop-1);

%-----------------------------------------------------

%                                          Print output of values on screen
fprintf('-------------------------------------------\n')
fprintf('                                   Calculation values
                \n')
fprintf('The Courant number is.....%1.2f\n',CFL)
fprintf('The x distance to calculate over is.....%3.2f m\n',h)
fprintf('The y distance to calculate over is.....%3.2f m\n',h)
fprintf('The number of x grid points is.....%4.0f\n',stop-1)
fprintf('The number of y grid points is.....%4.0f\n',stop-1)
fprintf('The x step size is.....%1.4f m\n',dx)
fprintf('The y step size is.....%1.4f m\n',dx)
fprintf('-------------------------------------------\n')
fprintf('**********************************************************************
*\n')
fprintf('                          Conditions ahead of the shock
\n')

fprintf('Gamma is.....%1.1f\n',g)
fprintf('The initial pressure is.....%3.3g Pa\n',Pin)
fprintf('The initial density is.....%1.4f kg/m^3\n',rho)
fprintf('The initial temperature is.....%3.3g K\n',T1)
fprintf('The speed of sound is.....%3.4f m/s\n',c)
fprintf('**********************************************************************
*\n')

%-----------------------------------------------------
```

```
endt=input('Enter the number of time steps to calculate over: ');
                if endt<=0
                error('You must enter a number of time steps larger than zero!')
                end

%-------------------------------------------------------------
%                                                  define vector size

for             x=1:stop
for             y=1:stop
op(x,y)=0.0;
np(x,y)=0.0;
Olr(x,y)=0.0;
nr(x,y)=0.0;
br(x,y)=0.0;
Olru(x,y)=0.0;
nru(x,y)=0.0;
Olrv(x,y)=0.0;
nrv(x,y)=0.0;
bru(x,y)=0.0;
nu(x,y)=0.0;
brv(x,y)=0.0;
nv(x,y)=0.0;
oE(x,y)=0.0;
nE(x,y)=0.0;
bE(x,y)=0.0;


end
end

%-------------------------------------------------------------
%               Calculate Peak static overpressure Ps based on charge
%                size and range to charge the driving force for the
%                calculations to follow
%-------------------------------------------------------------
%                                  r=range to charge
                (m)
%-------------------------------------------------------------

r=input('Enter the range to the charge in meters: ');
                if r<=0
                error('You must enter a range larger than zero!')
```

```
                end

%               Print out table of explosive types
%                               T=type of explosive
fprintf('-------------------------------------------------\n')
fprintf('       Choose the explosive using the table below\t\n')
fprintf('Compound B .....1\n')
fprintf('RDX (Cyclonite).....2\n')
fprintf('HMX .....3\n')
fprintf('Nitroglycerin (liquid).....4\n')
fprintf('TNT .....5\n')
fprintf('Blasting Gelatin .....6\n')
fprintf('60 percent Nitroglycerin dynamite .....7\n')
fprintf('Semtex .....8\n')
fprintf('-------------------------------------------------\n')

%---------------------------------------------------------
%               Ask for what type of explosive to use
%---------------------------------------------------------

T=input('enter the type of explosive to use: ');
                if T==1
                                S=1.148;
                elseif T==2
                                S=1.185;
                elseif T==3
                                S=1.256;
                elseif T==4
                                S=1.481;
                elseif T==5
                                S=1.000;
                elseif T==6
                                S=1.000;
                elseif T==7
                                S=0.600;
                elseif T==8
                                S=1.250;
                else
                error('You must enter an explosive type!')
                end
fprintf('The TNT conversion factor for this explosive is: %1.3f\n',S)

%---------------------------------------------------------
```

```
%               Ask for mass of charge
                (kg)
%---------------------------------------------------------

M=input('Enter the mass of the charge in kilograms: ');
                if M<0
                error('You must enter a mass larger than zero!')
                end
W=M*S;

%---------------------------------------------------------
%                               Calculate peak static overpressure              (B)
%---------------------------------------------------------

z=r/(W^(1/3));
Ps=(6.7/(z^3));
                if Ps<10
                                Ps=((0.975/z)+(1.455/(z^2))+(5.85/(z^3)))-0.019;
                else Ps=(6.7/(z^3));
                end

                if Ps<0.1
                                Ps=Ps;
                end

%---------------------------------------------------------
%               Convert Ps(the static overpressure) from Bar (Pa)
%                to Pascals from overpressure to absolute
%                pressureand apply conversion factor for
%                ground burst
%---------------------------------------------------------

Ps=1.8*((Ps*100000)+Pin);


%---------------------------------------------------------
%                                               Print output of values on screen
%---------------------------------------------------------

fprintf('-------------------------------------------------------------\n')
fprintf('                                       Explosive values
                \n')
fprintf('The range to the charge from the x=0 point is.....%4.2f m\n',r)
```

```
fprintf('The mass of the charge in TNT equivalents is.....%4.3f kg\n',W)
fprintf('The static overpressure of the charge is.....%1.3g Pa\n',Ps-Pin)
fprintf('-----\n')

%-----------------------------------------------------
%                                   Calculate intial velocity using Ps
%-----------------------------------------------------

Mach=sqrt(((Ps/Pin)-1)*((g+1)/(2*g))+1);
Vin=c*(2/(g+1)*(Mach-(1/Mach)));
%                                   Print output of values on screen
fprintf('The initial wave velocity is.....%4.4f m/s\n',Vin)
fprintf('The shock wave Mach number is.....%2.4f\n',Mach)
fprintf('-----\n')

%-----------------------------------------------------
%                                   Calculate values behind the shock
%-----------------------------------------------------

rho2=rho/(1-(2/(g+1))*(1-(1/Mach^2)));
T2=Ps/(rho2*(8314.51/29));
c2=(c^2*T2/T1)^0.5;
fprintf('**********************************************************************
*\n')
fprintf('                          Conditions behind the shock
\n')
fprintf('The pressure is.....%1.3g Pa\n',Ps)
fprintf('The density is.....%1.4f kg/m^3\n',rho2)
fprintf('The temperature is.....%3.3g K\n',T2)
fprintf('The speed of sound is.....%3.4f m/s\n',c2)
fprintf('**********************************************************************
*\n')
%-----------------------------------------------------

%-----------------------------------------------------
%                                   Define initial conditions
%-----------------------------------------------------

for         x=1:stop
for         y=1:stop
            op(x,y)=Pin;
            Olr(x,y)=rho;
            Olru(x,y)=0;
```

```
                Olrv(x,y)=0;
                oE(x,y)=Pin/g1;
end

end

%-------------------------------------------------------
%               Define boundary conditions at grid point x=1 y=1
%-------------------------------------------------------
                y=2;
                x=2;
                op(x,y)=Ps;
                Olr(x,y)=rho/(1-(2/(g+1))*(1-(1/Mach^2)));
                Olru(x,y)=Olr(x,y)*Vin;
                Olrv(x,y)=Olr(x,y)*Vin;
                oE(x,y)=(Ps/g1)+(0.5*(Olru(x,y)^2+Olrv(x,y)^2)/...
                  Olr(x,y));



%-------------------------------------------------------
%               Calculate the new speed of sound based on new pressure
%                and density values
%-------------------------------------------------------

                nc=sqrt(g*op(x,y)/Olr(x,y));
%-------------------------------------------------------
%                                       Calculate the initial time step
%-------------------------------------------------------

    dt=CFL*(1/(abs(Olru(1,1)/Olr(1,1))/dx+abs(Olrv(1,1)/Olr(1,1))/...
                  dy+nc*(1/dx^2+1/dy^2)^(0.5)));



fprintf('The inital time step is.....%1.5e s\n',dt)
%-------------------------------------------------------
%                                       Initialize total time counter
%-------------------------------------------------------

tt=0;
```

```
%-----------------------------------------------------
%               Finite difference for density, momentum and energy
%-----------------------------------------------------

%-----------------------------------------------------
%               Adjust x and y grid end point for finite differencing
%-----------------------------------------------------

stop=stop-1;



for t=1:endt

%-----------------------------------------------------
%               NBS for column 1 and row 1
%-----------------------------------------------------

               dtmin=1.;

               dtdx=dt/dx;
               dtdy=dt/dy;

%-----------------------------------------------------
%                                          Calculate total time
%-----------------------------------------------------

               tt=tt+dt;


%-----------------------------------------------------
               for x=1:stop
               for y=1:stop

%-----------------------------------------------------


%                                                    PREDICTOR
%-----------------------------------------------------
%               Solve as a function of time and position for
%                increasing time
```

```
%-------------------------------------------------------
%                                                Density(r)
%-------------------------------------------------------

            br(x,y)=Olr(x,y)-...
   (dt*(((Olru(x+1,y)-Olru(x,y))/dx)+...
               ((Olrv(x,y+1)-Olrv(x,y))/dy)));

            br(x,stop+1)=br(x,stop);
            br(stop+1,y)=br(stop,y);
            br(stop+1,stop+1)=br(stop,stop);
%-------------------------------------------------------
%                                              Momentum(ru)
%-------------------------------------------------------

            bru(x,y)=Olru(x,y)-dt*...
              ((1/dx*...
              (Olru(x+1,y)^2/Olr(x+1,y)-...
               Olru(x,y)^2  /Olr(x,y)  +...
              (op(x+1,y)-op(x,y)      )))+...
              (1/dy*...
              (Olrv(x,y+1)*Olru(x,y+1)/Olr(x,y+1)-...
               Olrv(x,y)  *Olru(x,y)  /Olr(x,y)  )));

            bru(x,stop+1)=bru(x,stop);
            bru(stop+1,y)=bru(stop,y);
            bru(stop+1,stop+1)=bru(stop,stop);

%-------------------------------------------------------
%                                              Momentum(rv)
%-------------------------------------------------------

            brv(x,y)=Olrv(x,y)-dt*...
              ((1/dx*...
              (Olru(x+1,y)*Olrv(x+1,y)/Olr(x+1,y)-...
               Olru(x,y)  *Olrv(x,y)  /Olr(x,y)  ))+...
              (1/dy*...
              (Olrv(x,y+1)^2/Olr(x,y+1)-...
               Olrv(x,y)^2  /Olr(x,y)+...
              (op(x,y+1)-op(x,y)     ))));

            brv(stop+1,y)=brv(stop,y);
            brv(x,stop+1)=brv(x,stop);
```

```
            brv(stop+1,stop+1)=brv(stop,stop);
%-------------------------------------------------------
%                                                Energy(e)
%-------------------------------------------------------

            bE(x,y)=oE(x,y)-dt*...
              ((1/dx*...
              ((oE(x+1,y)+op(x+1,y))*Olru(x+1,y)/Olr(x+1,y)  -...
               (oE(x,y)+op(x,y)   )*Olru(x,y)  /Olr(x,y)   )+...
                1/dy*...
              ((oE(x,y+1)+op(x,y+1))*Olrv(x,y+1)/Olr(x,y+1)  -...
               (oE(x,y)+op(x,y)   )*Olrv(x,y)  /Olr(x,y)   )));

            bE(stop+1,y)=bE(stop,y);
            bE(x,stop+1)=bE(x,stop);
            bE(stop+1,stop+1)=bE(stop,stop);
%-------------------------------------------------------

% return for next x
end
% return for next y
end

%-------------------------------------------------------

for         x=2:stop+1
for         y=2:stop+1

%-------------------------------------------------------
%                                                CORRECTOR
%-------------------------------------------------------
%                                                Density(r)
%-------------------------------------------------------

            nr(x,y)=(br(x,y)+Olr(x,y))/2-...
              ( dtdx/2*(bru(x,y)-bru(x-1,y))+...
                dtdy/2*(brv(x,y)-brv(x,y-1)));



            nr(1,y)=nr(2,y);
            nr(x,1)=nr(x,2);
            nr(2,2)=Olr(2,2);
```

```
                nr(1,1)=Olr(2,2);

%               Left wall boundary conditions

%-------------------------------------------------------

%                                                   Momentum(ru)
%-------------------------------------------------------
 Px=g1*(bE(x,y)-0.5*((bru(x,y)^2/br(x,y))+(brv(x,y)^2/br(x,y))));
 Pxm=g1*(bE(x-1,y)-0.5*((bru(x-1,y)^2/br(x-1,y))+(brv(x-1,y)^2/br(x-1,y))));

                nru(x,y)=(bru(x,y)+Olru(x,y))/2-...
                  ((dtdx/2*...
                  (( bru(x,y)  *bru(x,y)  /br(x,y)  +Px )-...
                   ( bru(x-1,y)*bru(x-1,y)/br(x-1,y)+Pxm)))+...
                  (dtdy/2*...
                  (  bru(x,y)  *brv(x,y)  /br(x,y)-...
                    bru(x,y-1)*brv(x,y-1)/br(x,y-1)    ))) ;

%-------------------------------------------------------


                nru(1,y)=nru(2,y);
                nru(x,1)=nru(x,2);
                nru(2,2)=Olru(2,2);
                nru(1,1)=Olru(2,2);

%                                                   Momentum(rv)
%-------------------------------------------------------

 Py=g1*(bE(x,y)-0.5*((bru(x,y)^2/br(x,y))+(brv(x,y)^2/br(x,y))));
 Pym=g1*(bE(x,y-1)-0.5*((bru(x,y-1)^2/br(x,y-1))+(brv(x,y-1)^2/br(x,y-1))));

                nrv(x,y)=(brv(x,y)+Olrv(x,y))/2-...
                  ((dtdx/2*...
                  ( bru(x,y)  *brv(x,y)  /br(x,y)    -...
                    bru(x-1,y)*brv(x-1,y)/br(x-1,y) ))+...
                  (dtdy/2*...
                  ( (brv(x,y)*brv(x,y)/br(x,y)+    Py  )-...
                    (brv(x,y-1)*brv(x,y-1)/br(x,y-1)+Pym))));

%-------------------------------------------------------
```

```
                nrv(1,y)=nrv(2,y);
                nrv(x,1)=nrv(x,2);
                nrv(2,2)=Olrv(2,2);
                nrv(1,1)=Olrv(2,2);

%               Left wall boundary conditions

%---------------------------------------------------------

%                                                       Energy(E)
%---------------------------------------------------------

                nE(x,y)=(bE(x,y)+oE(x,y))/2-...
                  ((dtdx/2*...
                  ((bru(x,y) /br(x,y) *(bE(x,y) +Px )) -...
                  (bru(x-1,y)/br(x-1,y)*(bE(x-1,y)+Pxm)))) +...
                  (dtdy/2*...
                  ((brv(x,y) /br(x,y) *(bE(x,y)+Py )) -...
                  (brv(x,y-1)/br(x,y-1)*(bE(x,y-1)+Pym)))) );
%---------------------------------------------------------
%---------------------------------------------------------


                nE(1,y)=nE(2,y);
                nE(x,1)=nE(x,2);
                nE(2,2)=oE(2,2);
                nE(1,1)=oE(2,2);
%---------------------------------------------------------
%return for next x
                end
%---------------------------------------------------------
%return for next y
                end
%---------------------------------------------------------
for             y=1:stop+1
for             x=1:stop+1
%---------------------------------------------------------
%               Calculate pressure using the continuity equation for
%               the ideal gas law and the new values for energy,
%               momentum and density
%---------------------------------------------------------
```

```
np(x,y)=(g1)*(nE(x,y)-(.5*((nru(x,y)^2/...
  nr(x,y))+(nrv(x,y)^2/nr(x,y)))));



%-------------------------------------------------------
%           Calculate velocity using the new values for momentum
%            and density
%-------------------------------------------------------

           nu(x,y)=nru(x,y)/nr(x,y);
           nv(x,y)=nrv(x,y)/nr(x,y);


%-------------------------------------------------------
%                                                  Check stability
%                                        Calculate the new speed of sound
%-------------------------------------------------------

           nc=sqrt(g*np(x,y)/nr(x,y));
%-------------------------------------------------------
%
%                                        Calculate the new time step dt
%-------------------------------------------------------


           dtn=CFL*(1/(abs(nu(x,y))/dx+abs(nv(x,y))/dy+...
             nc*(1/dx^2+1/dy^2)^(0.5)));
           if dtn < dtmin
                        dtmin=dtn;
% end dtmin if statement
           end
%-------------------------------------------------------

%-------------------------------------------------------

%-------------------------------------------------------
%return for next x
           end
%-------------------------------------------------------
%return for next y
           end
```

```
%-----------------------------------------------------------


%-----------------------------------------------------------
%               Update time step and CFL(Courant number)
%-----------------------------------------------------------

    dt=dtmin;

fprintf('delta t= %1.5e s\tTotal time= %1.5e s\n',dt,tt)



%-----------------------------------------------------------
%               Output the values for pressure, density and velocity
%                using graphs
%-----------------------------------------------------------

%               Pressure
for             x=2:stop+1
for             y=2:stop+1
                press(x-1,y-1)=np(x,y);
end
end

colormap(cool)
surf(press)
hold on
pcolor((press)-Pin)
%contour(press,20)
title('Absolute Pressure')
xlabel('X grid points')
ylabel('Y grid points')
zlabel('Pascals')

pause(.01)
%hold off
%-----------------------------------------------------------
%               Replace old values with new values and begin
%                next time step
%-----------------------------------------------------------

op=np;
```

```
Olr=nr;
Olru=nru;
Olrv=nrv;
oE=nE;
%-------------------------------------------------------------
%                Return for next t
end
%-------------------------------------------------------------
```